PDP-11 Software Handbook

digital

PDP-11 Software Handbook

Contents

Preface

The *PDP-11 Software Handbook* provides a description of all the PDP-11 operating systems, utilities, and languages available from Digital.

Chapter 1 provides a concise overview of the various operating systems offered for use with Digital's PDP-11 computers.

Chapter 2 covers general information about an operating system and provides a technical, yet easy-to-understand, description of its components.

Chapter 3 provides a detailed description of the *RSX* family of realtime, multiprogramming operating systems.

Chapter 4 provides information about *RSTS/E*, a proven, versatile timesharing system.

Chapter 5 covers the *RT-11* system, a compact operating system for realtime, single-user applications, and the *CTS-300* operating system, a layered product that combines the RT-11 system and the DIBOL language for multiuser, commercial timesharing applications.

Chapter 6 discusses *DSM* (Digital Standard MUMPS), a multiuser data management timesharing system.

Chapter 7 provides information about *MicroPower/Pascal*, a software tool kit for developing PDP-11 (Q-bus) based microcomputer applications.

Chapter 8 discusses the features and benefits of *ULTRIX-11*, Digital's version of the UNIX™ operating system.

Chapter 9 gives an overview of the *IAS* operating system, a mature multiuser timesharing operating system.

Chapter 10 gives a brief introduction to the Digital-supplied *programming languages* that can be used to meet specific application needs.

Chapter 11 explains the features and benefits of the *MACRO* assembly-level language.

Chapter 12 provides a description of the *FORTRAN* language with applications in realtime control, computation, and general data processing.

Chapter 13 covers Digital's *BASIC* language, an easy-to-learn, conversational programming language.

Chapter 14 describes the *COBOL* language, a high-level, industrywide data processing language designed for business applications.

*UNIX is a trademark of AT & T Bell Laboratories.

Chapter 15 includes a description of *DIBOL*, Digital's business-oriented program development language.

Chapter 16 discusses the *PASCAL/RSX* language, a high-level, structured programming language.

Chapter 17 describes a number of *File Management Utilities* designed to make file management easier.

Chapter 18 describes *RMS*, a set of general purpose file-handling capabilities that provides efficient and flexible data storage and modification.

Chapter 19 covers *DATATRIEVE*, an interactive, query report generation and data maintenance system.

Chapter 20 discusses *EDT*, an easy-to-learn text editor that provides a wide range of online editing and file creation capabilities.

Chapter 21 provides information about *Screen Formatters*, the software tools that provide a forms management system.

Chapter 22 describes the *Networking and Data Communications* software that allow system-to-system communication.

Chapter 23 describes the *A-to-Z Integrated System*, a software package designed to provide business graphics, word processing, electronic mail, and other office solutions.

Chapter 24 provides information on *Cross-system Coexistence and Migration Tools* that allow communications between PDP-11 systems and other systems.

Appendix A summarizes the *Customer Services Programs* available for PDP-11 systems including Field Service, Software Services, and Education Services.

Appendix B introduces you to *DECUS*, the Digital Equipment Computer Users Society. DECUS helps its members obtain specialized software developed by other members.

Appendix C gives a brief summary of the *PDP-11 Software Source Book* that describes over 2,000 applications available for PDP-11 users.

Appendix D lists additional documentation and publications available for PDP-11 family hardware and software.

Appendix E contains a list of commonly used abbreviations found in this handbook.

Appendix F contains a listing of ASCII Codes to be used as a source of reference.

A glossary of software terms and an index are also included for your convenience.

Digital is constantly improving existing products and introducing new ones. Your best source for the most current information about software is your Digital sales representative, who can keep you up-to-date on recent releases and enhancements of these products.

A postage-paid Reader's Comment card is located at the back of this handbook. We hope you will take the time to complete it. Your suggestions will help us to continue to provide literature that meets your needs.

Chapter 1 • PDP-11 Software Overview

▪ Integration and Compatibility Are Key Features of the PDP-11 Family

The PDP-11 computer family is a wide range of compatible processors complemented by a variety of peripheral devices, software, and services. The PDP-11 computers are based on a 16-bit architecture to provide both low-cost and high-performance features.

This handbook discusses the major features and benefits of Digital's software products available for the PDP-11 family of computers. Software is the collection of programs or routines that allow people to use computer hardware. Generally speaking, programming creates and changes software.

Because your computer hardware is only as useful as the software operating with it, you should understand certain software concepts. This chapter and the next introduce some basic software concepts and shows how Digital implements those ideas.

Integration allows you to develop a consistent computing environment in which applications and computer resources can be added without interfering with existing operations, obsoleting existing equipment, or creating the need to retrain people. When computer resources need to be expanded, you can choose from the PDP-11 family of computers or easily migrate to a VAX system. DECnet, one of Digital's networking products can link Digital systems together to make the exchange of information quick and efficient. If you need to communicate with other manufacturers' mainframes, personal computers, or workstations, you can integrate between systems using Digital's many interconnection and networking products.

Integration of computer resources also means that you have the flexibility to develop applications on PDP-11 operating systems and migrate them to VAX systems and other systems via gateways. Within the PDP-11 family of computers are proven, supportable application tools to help expedite the development of new applications and solutions. One of Digital's widely used application tools is called *A-to-Z Software*. A-to-Z Software provides a set of generic layered applications including word processing, graphics, electronic mail, and report generating tools that are completely integrated with one another.

Compatibility is a key feature of both hardware and software in the PDP-11 world. Small systems can grow easily into larger ones as your data processing needs increase, without the heavy reinvestment required in starting from scratch. Your software can run in different PDP-11 hardware environments. For example, most operating systems can run on a wide range of processors, and all can grow to accommodate new languages, peripherals, and other applications.

While a few of the characteristics of software may vary from application to application, compatibility helps guarantee that personnel and programs can move among systems with a minimum of trouble.

The FORTRAN-77 programming language, for example, runs on most Digital PDP-11 operating systems. Once you've learned FORTRAN-77 on one system, you could, with little difficulty, write programs on another PDP-11 system running a different PDP-11 operating system. Likewise, a FORTRAN-77 application program can be readily transported to any Digital PDP-11 system that supports the language.

Most application software can also migrate with little difficulty across system families from the Professional series of personal computers to PDP-11 to VAX.

The flexibility of PDP-11 hardware and software allows you to select the most appropriate hardware, operating system, and languages to meet your immediate needs, while ensuring your system can grow whenever you need it to.

▪ Powerful Communications Networks Link PDP-11s

Digital produces powerful software and hardware that permits the linking of computers and terminals into flexible configurations called networks. Networks can vastly increase the efficiency and cost-effectiveness of data processing operations. No matter where your terminals or processors are located, they can be connected in ways that allow the exchange of information, files, programs, and control, as well as the sharing of peripherals. Small computers can access the powerful capabilities of mainframes when they are networked, while large computers can take advantage of smaller dedicated systems that have been chosen for specific application environments.

▪ Digital Offers a Wide Variety of Operating Systems for the PDP-11 Family

An operating system not only provides access to the features of a processor, it also organizes a processor and peripherals into a useful tool for a certain range of applications. Some systems can manage only one user's task at a time. Others accept many tasks. Some systems support realtime applications, that is, applications in which the computer is required to respond within given time limits to an external message, to make a decision, and to respond quickly (for example, flight simulator control, power plant management, and laboratory experiment supervision). Still others support timesharing applications in which the central processor is allocated according to a scheme of priorities, privileges, and time quotas (for example, airlines reservations systems and automated teller machines).

The operating systems featured in this handbook are

Micro/RSX	Packaging based on the multiuser, multitasking, RSX-11M-PLUS operating system and designed especially for the Micro/PDP-11.
RSX-11M-PLUS	Realtime System Executive Operating System-PLUS for high-end PDP-11 Processors A large realtime system designed to take advantage of enhanced hardware features and larger memory available on the PDP-11/44, MicroPDP-11/83, and PDP-11/84 processors. RSX-11M-PLUS is a superset of RSX-11M.
RSX-11M	Realtime System Executive Operating System for PDP-11 Processors A small-to-moderate-sized, realtime, multiprogramming system that can be generated for a wide range of application environments from small, dedicated systems to large, multipurpose, realtime application and program development systems.
RSX-11S	Realtime System Executive Operating System for PDP-11 Processors A small, execute-only member of the RSX-11 family for dedicated, realtime, multiprogramming applications (requires a host RSX-11M, RSX-11M-PLUS, IAS, or VAX/VMS system).

Micro/RSTS	Resource-Sharing Timesharing System/Extended Operating System for Micro PDP-11 Processors A general purpose, timesharing system that provides a fast response in multiuser applications and program development; designed to meet the needs of the MicroPDP-11 user.
RSTS/E	Resource-Sharing Timesharing System/Extended Operating System for PDP-11 Processors A moderate-to-large-sized timesharing system, expressly designed to optimize the interplay between people and system; RSTS/E is used in a variety of multiuser applications ranging from business data processing to academic instruction.
RT-11	Realtime Operating System for PDP-11 Processors A small, single-user, foreground/background system that is designed for realtime applications. Execution of realtime applications occur in the foreground along with any system jobs while the background is used for interactive program development or nontime-critical application execution.
CTS-300	Designed to support small business applications, CTS-300 is based on RT-11 and uses DIBOL (Digital's Business-Oriented Language), system utilities, and program development tools.
DSM-11	Digital-Standard MUMPS Operating System for PDP-11 Processors A small-to-large-sized timesharing system that offers a unique fast-access data storage and retrieval system for large database processing; originally designed for medical record management and now available for similar database applications.
MicroPower/Pascal	An advanced software tool kit that describes two system environments for developing microcomputer applications: a host system that you use to create, build, and test realtime application software, and a target minicomputer system that runs the software.

ULTRIX-11	A simple, elegant, easy-to-use interactive programming environment for multiple users based on an enhanced version of the UNIX™ Timesharing System Edition 7, developed by AT&T Bell Laboratories. ULTRIX is also system 5.2 compatible.
IAS	Interactive Application System This traditional product is a large, multiuser timesharing system that allows realtime application execution concurrently with timeshared interactive and batched processing.

Chapter 2 of this handbook defines and illustrates the concept of an operating system.

PDP-11 Operating Systems Support a Variety of Software Products—Bundled and Unbundled

Some software products are packaged with other products and sold together as a unit. For example, the RSTS/E operating system includes a BASIC-PLUS language processor; MUMPS-11 language is inseparable from the DSM-11 operating system.

Separately sold products, on the other hand, are distinct options, not necessarily included as part of any other software. Special application software packages such as stress analysis, statistical analysis, or general accounting that you might use in your application are sold separately from the operating systems they work with.

The flexibility produced by this approach aids in tailoring a system to your specific needs, without sacrificing or omitting vital routines and utilities.

Digital's operating systems support a considerable variety of software products, both those that are packaged together and those that are sold separately, along with supporting numerous central processors and peripherals. The three tables that follow compare the processors on which PDP-11 operating systems run, the languages supported under each system, and the typical applications of each operating system.

Because both hardware and software are continually evolving and being improved, these tables are to be used as guides to currently available products.

*UNIX is a trademark of AT & T Bell Laboratories.

Table 1-1 ▪ Processors

Operating System	11/23	11/73	11/83	11/24	11/44	11/84
	Processors PDP-11 Family					
Micro/RSX	X	X	X			
RSX-11M-PLUS	X	X	X	X	X	X
RSX-11M	X	X	X	X	X	X
RSX-11S	X	X	X	X	X	X
Micro/RSTS	X	X	X			
RSTS/E	X	X	X	X	X	X
RT-11	X	X	X	X	X	X
DSM-11	X	X	X	X	X	X
MicroPower/ Pascal	X	X	X			
ULTRIX-11	X	X	X	X	X	X
IAS				X	X	X

Table 1-2 ▪ Operating System Summary

Feature	Micro/ RSX	RSX-11M-PLUS	RSX-11M	RSX-11S	Micro/ RSTS	RSTS/E	RT-11	CTS-300	DSM-11	Micro Power/ Pascal	ULTRIX-11	IAS
User Interface												
Shell											X	
DCL	X	X	X	X	X	X	X	X				X
MCR		X	X	X						X		X
CCL					X	X	X	X				
User-written	X	X	X				X	X				X
Text Editors												
Keypad	X	X	X		X	X	X	X				X
Line	X	X	X		X	X	X	X	X			X
Screen	X	X	X		X	X	X				X	X
Batch Processing	X	X	X		X	X	X	X				X

(cont.)

Table 1-2 ▪ Operating System Summary (Cont.)

Feature	Micro/ RSX	RSX-11M-PLUS	RSX-11M	RSX-11S	Micro/ RSTS	RSTS/E	RT-11	CTS-300	DSM-11	Micro Power/ Pascal	ULTRIX-11	IAS
File Management												
Multikey ISAM	X	X	X		X	X						X
Single-key ISAM	X	X	X		X	X		X				X
Sequential	X	X	X		X	X	X	X			X	X
Relative	X	X	X		X	X	X					
Random	X	X	X		X	X	X	X				X
Automatic Installation	X				X		X		X	X	X	X[1]

X[1] 90% Automatic Installation and 10% Manual Installation.

Table 1-3 ▪ Language Support

Languages	Micro/ RSX	RSX-11M-PLUS	RSX-11M	RSX-11S	Micro/ RSTS	RSTS/E	RT-11	DSM-11	Micro Power/ Pascal	ULTRIX-11	IAS
MACRO-11	X	X	X	X[1]	X	X	X	X	X		X
BASIC-PLUS					X	X	X				
BASIC-PLUS-2	X	X	X	X[1]	X	X					X
COBOL-81	X	X	X		X	X					
DIBOL-83	X	X				X	X[2]				
FORTRAN-77	X	X	X	X[1]	X	X	X				X
FORTRAN IV				X[1]		X	X			X	
PASCAL	X	X	X				X		X		
Standard MUMPS								X			
C										X	
Assembler										X	

X[1] Task Execution only.

X[2] Bundled as CTS-300.

Chapter 2 • Operating Systems

Operating Systems Manage Your Computer Needs

An operating system is a collection of control programs and routines designed to make computer hardware devices easy to use. Operating systems vary greatly in the kinds of hardware with which they work, in the range of complexity of tasks they handle, in the degree of adaptability to special user purposes, and in the programming languages that they support.

Operating systems provide a way by which a user's specific program(s) can run on the computer. In addition to controlling application programs, operating systems can provide utilities and routines to manage peripheral devices, to detect errors in programs, to keep user accounts, and to protect information.

Operating Systems Organize Your Work

Operating systems are collections of programs that organize a set of hardware devices into a working unit that people can use. Figure 2-1 illustrates the relationship between users, operating system, and hardware.

PDP-11 operating systems consist of two sets of software: the executive (or monitor) software and the system utilities.

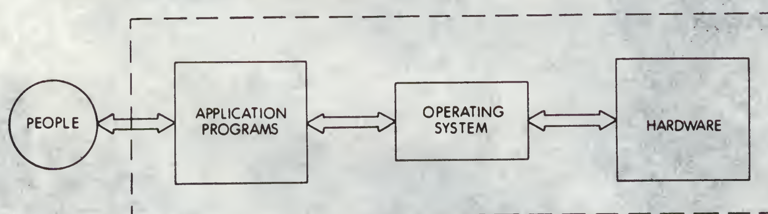


Figure 2-1 ▪ Computer System

An integrated set of routines, the executive acts as the primary interface between the hardware and a program running on the system, and between the hardware and the people who use the system. The executive's basic functions can be divided among the following services:

- User interface.
- Programmed processing services.
- Device and data management.
- Memory allocation.
- Processor time allocation.

In general, an executive can have two distinct operating components—a memory-resident portion and a temporarily resident (called transient) portion. Typically, the operating system is delivered on a hardware medium, called the system device, such as a disk or magnetic tape. In order for any useful work to be done, the executive must be transferred (loaded) into the memory part of the hardware processor.

When the executive is loaded into memory and started, all of it is resident. Its first duty is to communicate with the system operator. The executive simply waits until an operator requests some service, and then it performs that service. In general, such services include loading and starting programs, controlling program execution, modifying or retrieving system information, and setting system parameters. In most systems, these functions are serviced by transient portions of the executive. By being able to move back and forth between memory and the system device, the transient portion of the executive can make room for programs that need memory space, thereby improving the size-to-performance ratio of the computer. Movements of this sort occur in time spans measured in milliseconds.

The memory resident portion remains in memory to act on requests from the program, which generally include input/output (I/O) services such as file management, device-dependent operations, blocking and unblocking data, allocating storage space, and managing memory areas. In large systems, these services might also include intertask communication and coordination, memory protection, and task execution scheduling.

In some cases, the user can adjust the size of the executive by eliminating features that are not needed in an application environment. RSTS/E, RSX-11M, and RSX-11S are examples of such systems. RT-11 offers several executives of varying size and capability; the executive can be customized to add features. The RSX-11S system executive is always memory resident when the system is operating. In this case, the user concerned with size can remove routines that perform unneeded operations. In general, all PDP-11 operating systems are designed to be flexible enough to operate in a relatively wide range of hardware environments.

▪ System Utilities

System utilities are the individual programs supplied by Digital that are run under control of the executive to perform useful system-level operations. System utility programs enhance the capabilities of an operating system by providing users with commonly performed general services. There are three classes of system utilities—those used for program development; those used for file management; and those used to perform special system-management functions.

In the first category are the text editors, assemblers, compilers, linkers, program librarians, and debuggers. A whole section of this handbook is dedicated to programming languages and program development, and another section introduces some utilities such as editors and screen formatters that are useful in program development and in other contexts.

File management utilities include file copy, transfer, and deletion programs, file format translators, and media verification and clean-up programs. For more detailed information on these, see the handbook section on file and data management.

System management utilities vary from system to system, depending on the purpose and functions the system serves. Some examples are system information programs, user accounting programs, error logging, and online diagnostic programs.

Interactive Processing Lets You "Talk" to Your Computer

Interactive processing, as opposed to batch processing, permits dialogues between the computer and the user. People typing commands at terminals and getting quick response, and people writing and editing programs at video-terminals, are working interactively with the operating system. In batch processing, however, the whole program must be supplied, along with all necessary data, before any response can be obtained from the computer. Note that all Digital operating systems support interactive processing. Some support batch processing, as well.

The basic distinction among Digital operating systems is the processing method each uses to execute programs. The key distinctions among PDP-11 systems are

- Single-user vs. multiuser.
- Single-job vs. foreground/background.
- Foreground/background vs. multiprogramming.
- Timesharing vs. event-driven multiprogramming.

A single-user operating system receives demands upon its resources from a single source. It has only to manage the resources based on these demands. As a result, these systems do not require account numbers to access the system or data files. They usually don't protect the operating system from user programs. RT-11 is an example of a single-user operating system.

A multiuser operating system receives demands for its resources from many different individuals and/or programs. The system must manage its resources based on these demands. Several users may want sole control of a device at the same time, for example. The system handles access to the device. In addition, people may be using the system for different purposes, so some privacy must be maintained. As a result, a multiuser system normally has an account system to manage different user's files. RSTS/E, RSX-11M, and RSX-11M-PLUS systems are all multiuser systems, and all provide device allocation control and file accounts. In the case of the RSTS/E and RSX-11M systems, the file account structure is also used to keep track of the amounts of system resources an individual uses. Furthermore, the RSTS/E, RSX-11M, and RSX-11M-PLUS systems extend privacy by protecting individual users at a system level from the effects of any other users of the system.

An RT-11 system can operate in two modes — as a single-job system, or as a foreground/background system. In a foreground/background system, memory for user programs is divided into two separate regions. The foreground region is occupied by a program requiring fast response to its demands and priority on all resources while it is processing; a process-control or data acquisition application programs are examples of these "realtime" applications. The background region is available for a low-priority, preemptable program, for example, doing numerical analysis or program development.

Two independent programs, therefore, can reside in memory, one in the foreground region and one in the background region. The foreground program is given priority and executes until it relinquishes control to the background program. The background program is allowed to execute until the foreground program again requires control. Thus two programs effectively share the resources of the system.

When the foreground program is idle, the system does not go unused. Yet, when the foreground program requires service, it is immediately ready to execute. Input/output (I/O) operations, such as the input of data from the realtime process or the output of accounting files to the lineprinter, are processed independently of the requesting job to ensure that the processor is used efficiently as well as to enable fast response to all I/O interrupts.

The basis of foreground/background processing is the sharing of a system's resources between two tasks. An extension of foreground/background processing is multiprogramming. In multiprogrammed processing, many jobs compete for the system's resources. While it is still true that only one program can have control of the central processor at a time, concurrent execution of several tasks is achieved because other system resources, particularly I/O device operations printing text or waiting for input from a terminal, for example, can execute in parallel. While one task is waiting for an I/O operation to complete, another task can have control of the CPU. Time slicing (see below) can also manage multiprogramming environments.

RT-11 can also support systems of up to seven foreground programs and one background program, all scheduled by priority.

The RSX-11 family of operating systems employs multiprogrammed processing based on a priority-ordered queue of programs demanding system resources. In this case, memory is divided into several regions called partitions, and all tasks loaded in the partitions can execute in parallel. Program execution, as in the RT-11 foreground/background system, is event-driven. That is, a program retains control of the CPU until it declares a "significant" event normally meaning that it can no longer run, either because it has finished processing, or because it is waiting for another operation to occur. When a significant event is declared, the RSX-11 executive gives control of the CPU to the highest-priority task ready to execute. Furthermore, a high-priority task can interrupt a lower-priority task if it requires immediate service.

The RSTS/E and DSM systems also perform concurrent execution of many independent jobs. RSTS/E and DSM, however, process jobs on a timesharing rather than an event-driven basis, since timesharing is best suited for a purely interactive processing environment.

In a timesharing environment each job is guaranteed a certain amount of CPU time (a time slice). Jobs receive time one after another, in a round robin fashion. The system itself manages timesharing processing to obtain the best overall response, depending generally on whether jobs are compute-intense or I/O-intense. The system manager or privileged users can also specify the minimum guaranteed time for a particular job to service, as well as modify its priority.

▪ Operating Systems Can Be Tailored to Your Needs

System generation is the tailoring of an operating system to your particular hardware configuration and software services. Such structuring is necessary because each installation is unique, and each requires a different combination of the potential capabilities of a system. Your installation, for example, may not have a lineprinter, but may have extra disk drives or expanded memory. It would be wasteful to reserve valuable memory space for the code necessary to run lineprinters; also, you must configure the system so that it can use the disk drive. This is accomplished at system generation (SYSGEN) time.

System generation is also the point at which the system manager decides upon the inclusion, allocation, or definition of various utilities and system-wide parameters. In an RSX-11M system, for example, the manager could decide whether both event-driven and time-slice scheduling for realtime tasks should be available.

Finally, some layered software products are "added on" at SYSGEN time. (Other layered products might have been added after system installation, but before system generation.) If a RSTS/E manager wants to have the Pascal compiler available to users, for example, it is during system generation that the compiler is added.

Systems may, of course, be resysgened as needs change, as when the system grows, hardware is expanded, or additional compilers are added. Usually system generation is a routine procedure that involves a menu and a dialogue between the system and the manager or Digital software specialist. In some situations the SYSGEN can occur while the system is running; in others it may be necessary to bring the system down for the small time the operation requires.

▪ Data Management Manipulates Binary Information

Computers deal with binary information. Most people find it inconvenient to "think" in binary codes, so Digital's operating systems are programmed to translate easier-to-understand programming languages into binary. The way in which people interpret and manipulate the binary information is called *data management.*

This section describes PDP-11 software data management structures and techniques, from the physical storage and transfer level to the logical organization and processing level. (For definitions of words you are unfamiliar with, see the Glossary.) Contents of the following sections include

- ASCII and binary storage formats — how binary data can be interpreted.
- Physical and logical data structures — the difference between how data storage devices operate and how people use them.
- File structures — how physical units of data are logically organized for easy reference.
- File directories — how files are located and retrieved.
- File protection — how files are protected from unauthorized users.
- File naming conventions — how files are identified.

▪ Physical and Logical Units of Data

Computers — at their most fundamental level — understand only binary information. *Physical units* of data are the elements which computers use to store, transfer, and retrieve binary information. A *bit* (binary digit) is the smallest unit of data that computer systems handle.

In PDP-11 computers, a *byte* is the smallest memory-addressable unit of data. A byte consists of eight binary bits. An ASCII character code can be stored in one byte. Two bytes constitute a 16-bit *word*. Some machine instructions are stored in one word.

The smallest unit of data that a record-oriented I/O peripheral device can transfer is called its *physical record*. The size of a physical record is usually fixed and depends on the type of device being referenced. For example, a card reader can read and transfer 80 bytes of information at a time, stored on an 80-column punched card. The card reader's physical record length is thus 80 bytes. (Character-oriented devices — paper tapes and terminals — can obviously transfer a single character at one time.)

A *block* is the name for the physical record of a mass-storage device such as disk or magnetic tape. An RL02 disk block consists of 512 contiguous bytes. Its physical record length is 512 bytes.

Physical blocks can be grouped into a collection called a device or a physical *volume*. This collection has a size equal to the capacity of the device medium. The term physical volume is generally used with removable media, such as disk packs or magnetic tape.

Logical units of data are the elements manipulated by people and programs to store, transfer, and retrieve information. The information has logical characteristics, for example, data type (alphabetic or decimal) and size. The logical characteristics are not device-dependent; they are determined by the people using the system. It is the job of the operating system to correlate physical and logical data units. This frees programmers or users from worrying about manipulation, and allows them to concentrate on solving an application problem.

A *field* is the smallest logical unit of data. The field on a punched card, for example, used to contain a person's name is a logical unit of data. It can have any length necessary, determined by the programmer who defines the field.

A *logical record* is a collection of fields treated as a unit. It can contain any logically related information, in any one of several data types, and it can be any user-determined length. Its characteristics are not device-dependent, but they can be physically defined. For example, a logical record can occupy several blocks, or it can reside in a single block, or several logical records can reside in a single block. Its characteristics are determined by the programmer. A record, for example, could contain all of the status information about an item in inventory or about a loan applicant.

A *file* is a logical collection of data that occupies one or more blocks on a mass-storage device such as a disk or magnetic tape. A file is a system-recognized logical unit of data. Its characteristics can be determined by the system or the programmer.

A file can be a collection of logical records treated as a unit. An example is an employee file containing one logical record for each employee. Each record contains an employee's name and address and other pertinent information. If the logical record length is 50 bytes and there are 200 employees, the complete employee file could be stored in 20 512-byte blocks. Depending on the file structure used in the system, the blocks could be scattered over the disk, or could be located one after the other.

A *logical volume* is a collection of files that reside on a single disk or tape. It is the logical equivalent of a physical device unit (a physical volume) consisting of physical records, such as a disk pack. The files on a volume may have no specific relationship other than their residence on the same magnetic medium. In some cases, however, the files on a volume may all belong to the same user of the system.

Figure 2-2 illustrates some of the kinds of physical and logical units of data that PDP-11 computer systems handle.

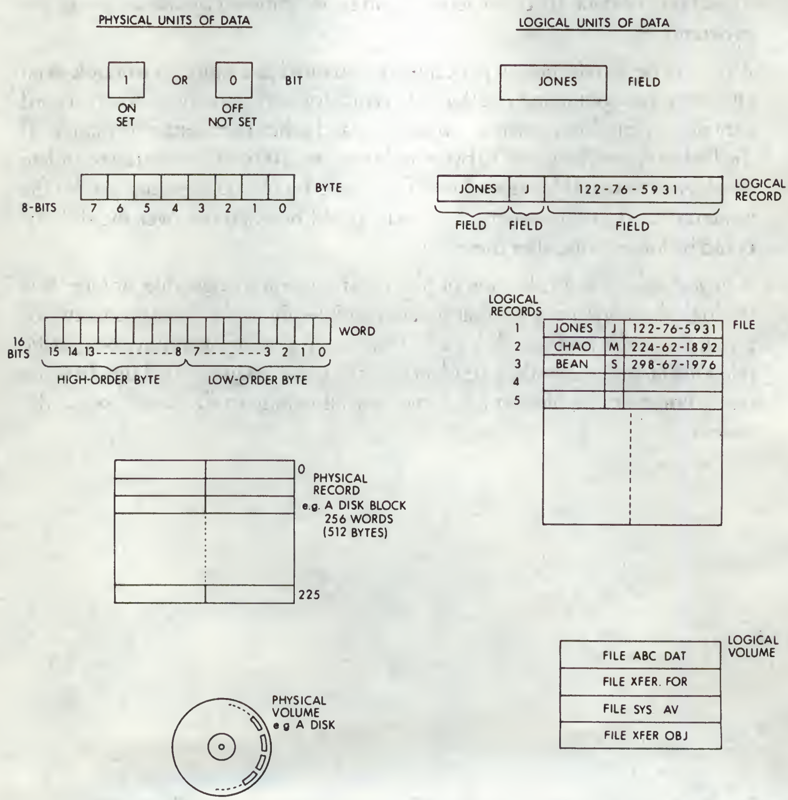


Figure 2-2 ▪ Physical and Logical Data Storage

In large, sophisticated systems such as RSTS/E, RSX-11M, and RSX-11M-PLUS, the way in which data are stored on the byte or bit level is rarely a concern of the application programmer. The operating system handles all data storage and transfer operations. In smaller systems such as RT-11, the programmer can become involved in data storage formats, although this is not generally a necessity. A particular application may require the selection of a particular storage format.

▪ Data Storage and Transfer Modes

All PDP-11 operating systems use two basic methods of data storage — ASCII and binary. Data stored in ASCII format conform to the American Standard Code for Information Interchange, in which each character is represented by a 7-bit code. The 7-bit code occupies the low-order seven bits of an 8-bit byte. The high-order bit is normally zero for PDP-11 systems. Text files are examples of data stored in ASCII format.

Binary storage always uses all eight bits of a byte to store information. The significance of any bit varies depending on the kind of information to be stored. Machine instructions (two's complement integer data) and floating-point numeric data are some examples of data stored in binary format.

Figure 2-3 illustrates the way in which binary data can be interpreted as either ASCII data or machine instructions. The figure shows examples of a word of storage containing a sequence of bits, interpreted first as two ASCII characters and second as a machine instruction.

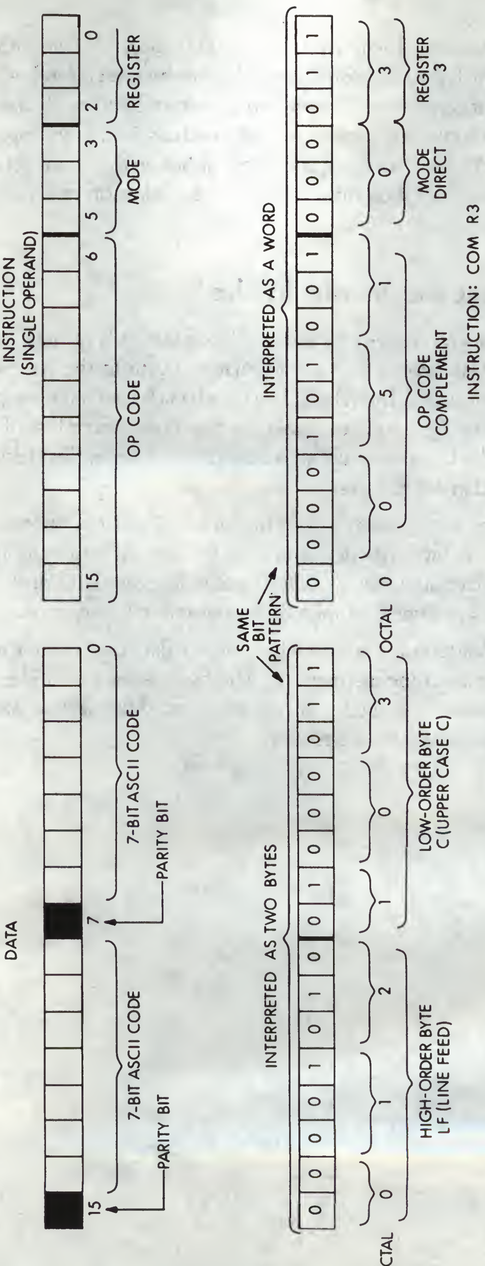


Figure 2-3 ▪ ASCII and Binary Storage

The data storage format is related to the way in which data are transferred in an I/O operation.

Formatting can also be applied at a higher level to define the type of data file being processed. In the RT-11 system, there are four types of binary files; each type signifies that a special interpretation applies to the kind of binary data stored. For example, a memory image file is an exact picture of what memory will look like when the file is loaded to be executed. A relocatable image file, however, is an executable program image whose instructions have been linked as if the base address were zero. When the file is loaded for execution, the system has to change all the instructions according to the offset from base address zero.

▪ I/O Devices and Physical Data Access Characteristics

In a PDP-11 computer system, data moves from external storage devices into memory, from memory into the CPU registers, and out again. The window from external devices to the CPU is called the I/O page. Each external I/O device in a computing system has an I/O page address assigned to it. Figure 2-4 illustrates the data movement path in a PDP-11 computing system.

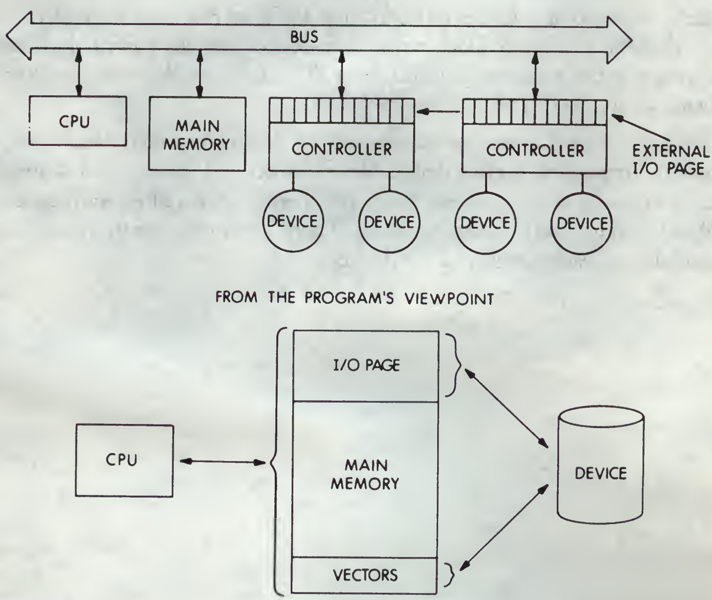


Figure 2-4 ▪ Memory and I/O Devices

Although all external devices are controlled similarly, devices differ in their ability to store, retrieve, or transfer data. Almost all PDP-11 operating systems provide device-independence among devices that have similar characteristics and, where possible, between differing devices in situations in which the data manipulation operations are functionally identical. Primarily, PDP-11 operating systems differentiate between

- File-structured and nonfile-structured devices.
- Block-replaceable and nonblock-replaceable devices.

Terminals and lineprinters are examples of devices that do not provide any means to store or retrieve physical records selectively. They can transfer data only in the sequence in which they occur physically.

In contrast, such mass-storage devices as disk and tape have the ability to store and retrieve physical records selectively. For example, an operating system can select a single file from among many stored on the medium.

Mass-storage devices are called file-structured devices because a file, consisting of a group of physical records, can be stored on and retrieved from the device. Terminals and lineprinters are called nonfile-structured devices because a file cannot be selectively read from or written to them.

Finally, mass-storage devices differ in their ability to read and write physical records. Disk devices are block-replaceable devices because a given block can be written without accessing or disturbing all the other blocks on the medium. Magnetic tape is not a block-replaceable device.

A device's physical data access characteristics determine which data transfer methods are possible for that device. Nonfile-structured devices allow sequential read or write operations only. Block-replaceable devices allow both sequential and random read or write operations. Figure 2-5 summarizes the read/write capabilities of each category of I/O device.

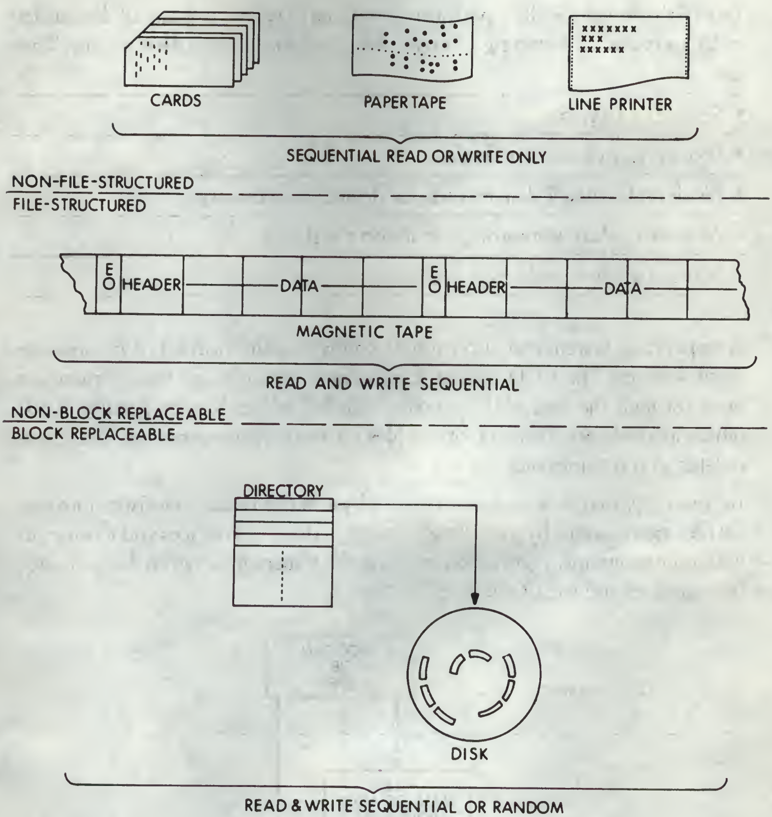


Figure 2-5 ▪ I/O Device Read/Write Capabilities

▪ Physical Device Characteristics and Logical Data Organizations

One of the most important services an operating system provides is the mapping of physical device characteristics into logical data organizations. You do not have to write the programs needed to handle input and output to any standard peripheral devices, since appropriate routines are supplied by Digital with the operating system.

There are generally two sets of routines provided in any operating system, depending on its complexity:

- Device drivers or handlers.
- File management services.

Device drivers or handlers perform operations to relieve the user of the burden of I/O services, overlapping I/O considerations, and device dependence. They can

- Service I/O devices.
- Provide device independence.
- Block and unblock data records for devices, if necessary.
- Allocate or deallocate storage space on the device.
- Manage memory buffers.

An operating system can also provide you with a uniform set of file management services. The RT-11 system, for example, provides file management services through the part of the monitor called the User Service Routine (USR), which loads device handlers, opens files for read/write operations, and closes, deletes, and renames files.

In summary, an operating system maps physical device characteristics into logical file organizations by providing routines to drive I/O devices and to interface with user programs. Figure 2-6 illustrates the transition between the user interface routines and the I/O devices.

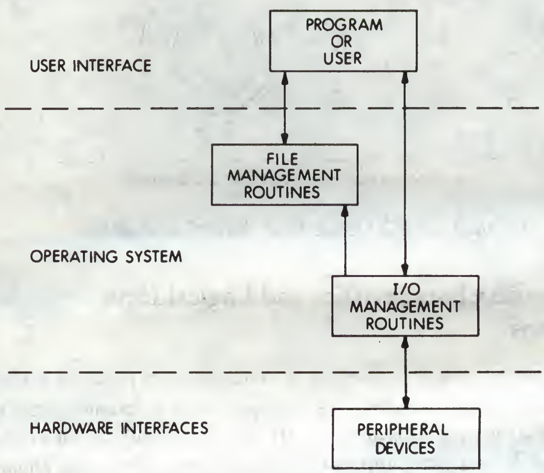


Figure 2-6 ▪ Device Control and File Management Services

As an example of the mapping of physical characteristics into logical organizations, the RSX-11 system's device driver and handler and file management services allow the user program to treat all file-structured devices in the same manner. That is, all of these devices appear to the user program to be organized into files consisting of consecutive 512-byte blocks that are numbered from block zero of the file to the last block of the file. In reality, the blocks may be scattered over the device and, in some cases, the device's actual physical record length may not be 512 bytes.

In RSX-11 terminology, the actual physical records on the device (for example, the sectors on a disk) are called physical blocks. At the device driver or handler level, the system maps these physical blocks into logical blocks. Logical blocks are numbered in the same relative way that physical blocks are numbered, starting sequentially at block zero — as the first block on the device — to the last block on the device. At the user interface level, the operating system maps logical blocks into virtual blocks. Virtual block numbers become file-relative values, while logical block numbers are volume-relative values.

Figure 2-7 illustrates the mapping between physical, logical, and virtual blocks in an RSX-11 system. The figure shows two disk device types that have different physical record lengths. In this case, the blocks constituting a file are scattered over the disk. The file is a total of five blocks long. At the logical block level, the operating system views the file as a set of noncontiguous blocks. At the virtual block level, the user software views the file as a set of contiguous, sequentially numbered blocks.

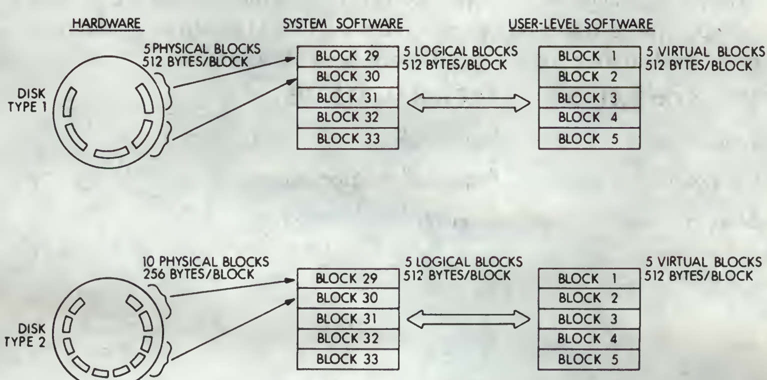


Figure 2-7 ▪ Physical, Logical, and Virtual Blocks

▪ File Structures and Access Methods

A file structure is a method of organizing logical records into files. It describes the relative physical locations of the blocks constituting a file. The file structure or structures that a particular operating system employs is a product of the way in which the system views the particular I/O devices and the kinds of data processing requirements the system fulfills.

File structure is important because a file can be effective in an application only if it meets specific requirements involving:

Size	Growth of the file may require a change in the file structure or repositioning of the file.
Activity	The need to access many different records in a file or frequently access the same file influences data retrieval efficiency.
Volatility	The number of additions or deletions made to a file may affect the access efficiency.

An *access method* is a set of rules for selecting logical records from a file. The simplest access method is sequential: each record is processed in the order in which it appears. Another common access method is direct access: any record can be named for the access. A nonblock-replaceable device, such as magnetic tape, can only be processed sequentially. A block-replaceable device, such as disk, can be processed by either access method, but direct access takes greatest advantage of the device's characteristics.

PDP-11 operating systems provide a variety of file structures and access methods appropriate to their processing services. All PDP-11 file structures are, however, based on some form of the following basic file structures:

FILE STRUCTURE	ACCESS METHODS
Linked	Sequential
Contiguous	Sequential or direct access
Mapped	Sequential or direct access

Linked files are a self-expanding series of blocks which are not physically adjacent to one another on the device. The operating system records data blocks for a linked file by skipping several blocks between each record. The system then has enough time to process one block while the medium moves to the next block to be used for recording. In order to connect the blocks, each block contains a pointer to the next block of the file. Figure 2-8a shows the format of a linked file.

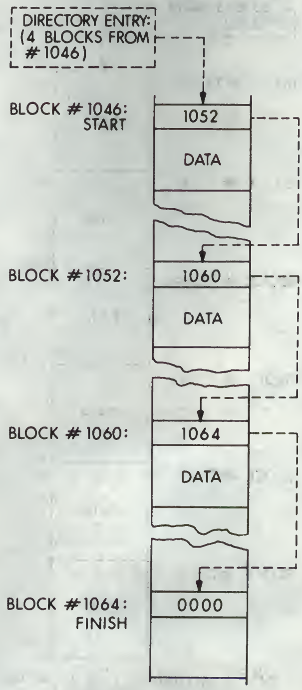


Figure 2-8a ▪ Linked File Structure

Linked file structure is especially suited for sequential processing where the final size of the file is not known. It readily allows later extension, because the user can add more blocks in the same way the file was created. In this way, linked files make efficient use of storage space. Linked files can also be joined together easily.

The blocks of *contiguous* files are physically adjacent on the recording medium. This format is especially suited for random (direct access) processing, because the order of the blocks is not relevant to the order in which the data is processed. The system can readily determine the physical location of a block without reference to any other blocks in the file. Figure 2-8b shows the format of a contiguous file.

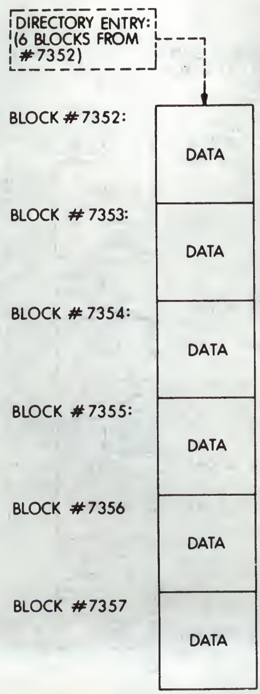


Figure 2-8b ▪ Contiguous File Structure

Mapped files are virtually contiguous files; they appear to the user program to be directly addressable sets of adjacent blocks. The files may not, however, actually occupy physically contiguous blocks on the device. The blocks can be scattered anywhere on the device. Separate information, called a file header block, is maintained to identify all the blocks constituting a file. This method provides an efficient use of available storage space and allows files to be extended easily, while still maintaining a uniform program interface. Figure 2-9a illustrates a mapped file format.

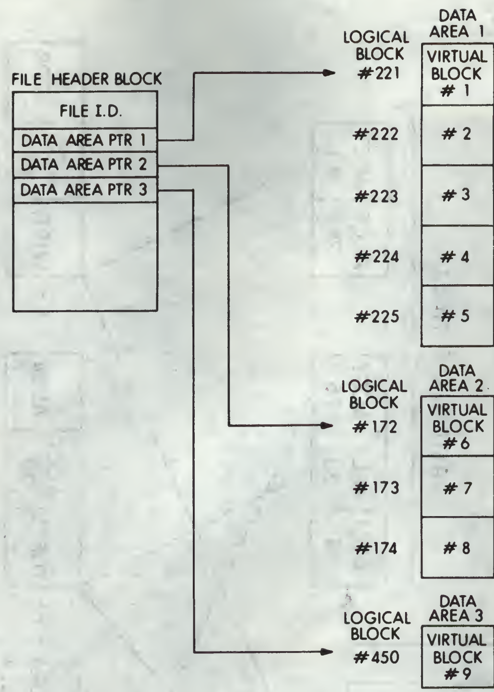


Figure 2-9a ▪ Mapped File Structure Non-contiguous File)

If desired, a mapped file can be created as a contiguous file to ensure the fastest random accessing, in which case it is both virtually and physically contiguous.

The basic file structures discussed above can be modified or combined to extend the features of each type for special-purpose logical processing methods. Some examples are indexed files and global array files.

For the most generalized and flexible file structure, you can use indexed files, which are actually two contiguous files. One file acts as an ordered map of a second file containing the target data. The index portion or map contains either an ordered list of key data selected from the target data records or pointers to data records in the second file, or both.

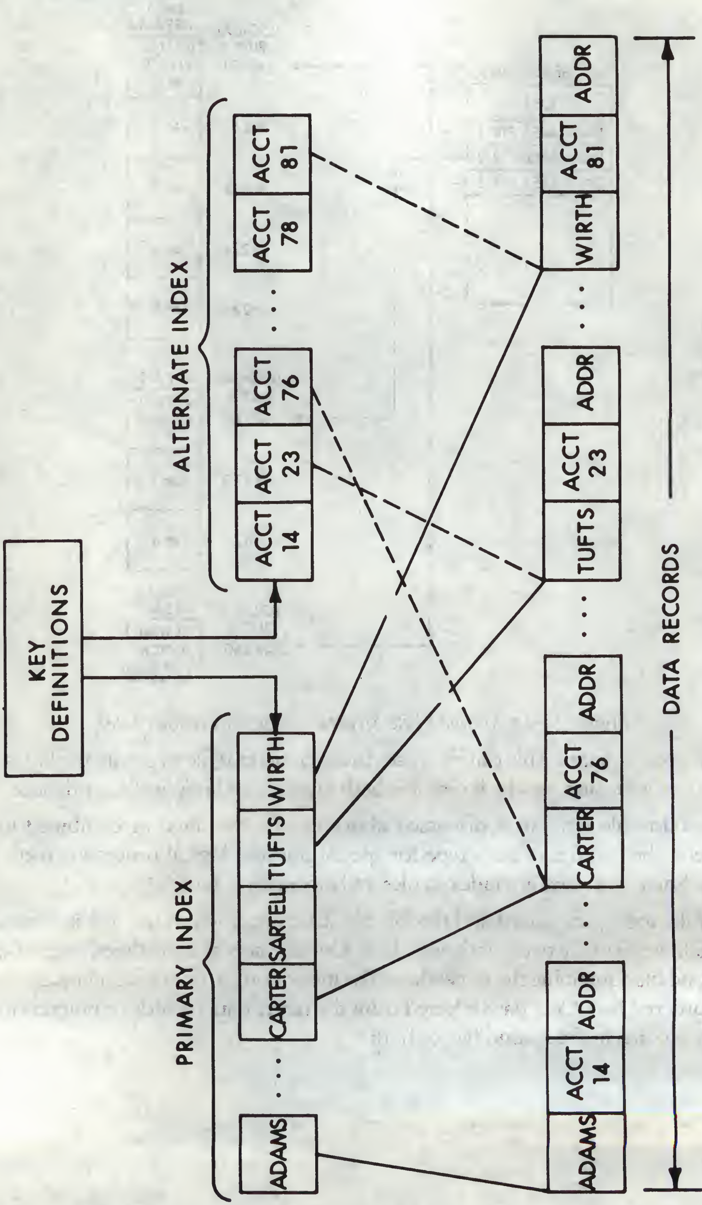


Figure 2-9b ▪ Indexed File Structure

The target data records can be processed in the order of the index portion, or the target data records can be selected by searching through the index portion for the key data identifying the records. These methods of logically processing the target data are called *indexed sequential access* and *random access by key*, respectively.

The Digital Standard Mumps (DSM) operating system provides another special file structure, called *global array files*, a version of the linked file structure. The arrays themselves are a logical tree-structured organization consisting of one or more subscripted levels of elements. All elements on a particular subscripting level are stored in a single chain of linked blocks. At the end of each block in the chain is a pointer to the next block in the chain. The levels of the array (all the block chains) are linked together through pointers in the first block of each chain. This file structure ensures that the time it takes to access any element of the array is minimal. Figure 2-9C shows the DSM global array structure.

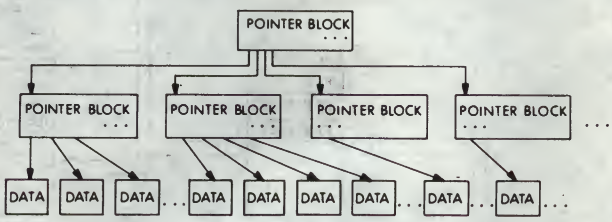


Figure 2-9c ▪ DSM Global Array Structure

▪ Directories and Directory Access Techniques

Just as file structure and access methods are required to locate records within files, directory structures and directory access techniques are required to locate files within volumes.

A directory is a system-maintained structure used to organize a volume into files. It allows the user to locate files without specifying the physical addresses of the files. It is a direct access method applied to the volume to locate files.

RT-11 supports the simplest kind of file directory. When disk and tape media are initialized for use, the system creates a directory on the device. Each time a file is created, an entry is made in the directory that identifies the name of the file, its location on the device, and its length. When access to the file is requested thereafter, the system examines the directory to find out where the file is actually located. The system can access the file quickly without having to examine the entire device.

In multiuser systems like RSTS/E and RSX-11M, the system uses two different kinds of directories to differentiate between files belonging to different users. They are the Master File Directory (MFD) and the User File Directories (UFD). These directories are maintained as files themselves, stored on the (physical) volume for which they provide a directory.

The MFD contains the names of all the possible users of a particular device. The UFD contains the names of all the files created by a particular user on a device. The system first checks the MFD to locate the UFD for the particular user, and then checks the UFD to locate the file. Figure 2-10 illustrates the use of the Master File Directory and the User File Directory.

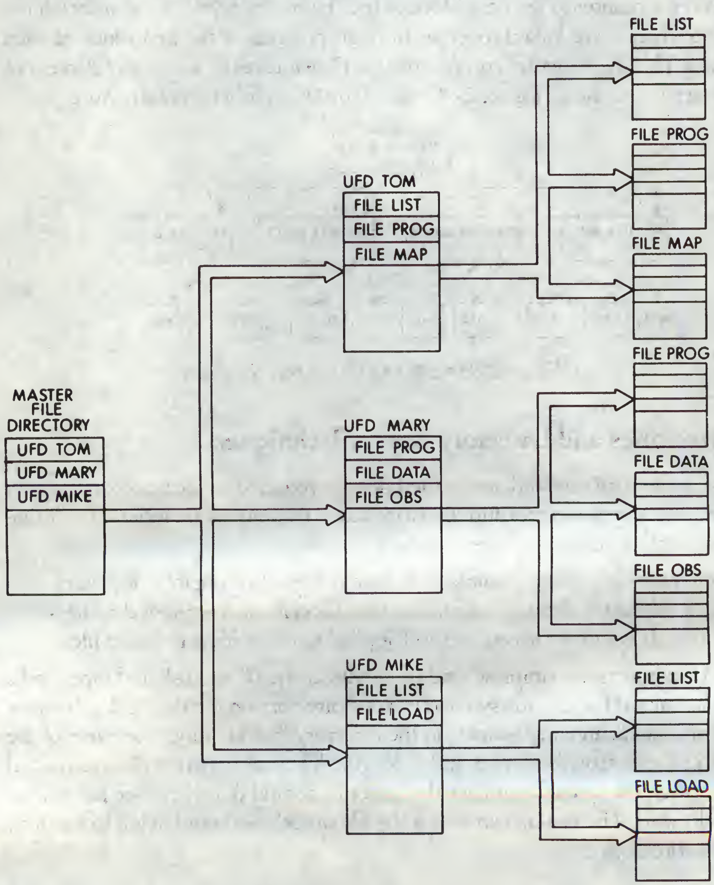


Figure 2-10 ▪ Master and User File Directories

Though the directories are important in the operation and access of the system, most users need not worry about them. The system maintains directories on behalf of all users. Of course, as with most system services, privileged users may do their own directory creation and maintenance.

▪ File Protection

RT-11 provides the simplest form of file protection with a one-bit "protected" designation. Files so named cannot be accidentally deleted. Under RT-11 no system of user numbers or accounts is needed.

Directories form the basis for file access protection in multiuser systems. Unauthorized users cannot access a file unless they know the account under which it is stored and can obtain access to that account. Account systems and file access protection techniques are related.

Multiuser systems identify the individuals who use the system by account numbers called User Identification Codes (UICs), which are normally assigned by the system manager. In general, a UIC consists of two numbers: the first number is used to identify a related group of users and the second number is used to identify an individual user in the group.

The RSX-11 file system provides a protection scheme for both volumes and files. You can specify protection attributes for an entire volume as well as for the files in the volume. A file or an entire volume can be read-, write-, extend- or delete-protected. Distinctions are made on the basis of account number, where the system recognizes four groups of users: privileged users, owner, owner's group, and all others.

In RSTS/E systems, an individual file can be protected against read access or write access where distinctions are made on the basis of the UIC account number under which a file is stored. A file can be read-protected, for example, against all users who are not in the same account group and write-protected against all users except the owner.

▪ File Naming

The most common way users communicate their desire to process data is through file specifications. A file specification uniquely identifies and locates any logical collection of data that is online to a computer system.

A language processor, for example, needs to know the name and location of the programming language source program file that it is to compile; it also needs to know the name that the user wants to use for the output object program and listing files it produces. Most PDP-11 operating systems share the same basic format for input and output file specifications.

Typically, the file specification includes a device name (given by an abbreviation mnemonic) for the device where the file resides, a unit number, the file name itself, and a file type — a group of one to three characters that conventionally tells what kind of file it is. (For example, .FOR as a file type says the file is a source program in the FORTRAN language.) In multiuser systems, the file specification might also include the UIC of the user and the file version number. If a network application is being run, the file specification would also include the node name of the node where the file is, if other than the host system.

In most cases, the user does not have to issue a complete file specification. The PDP-11 operating systems use default values when a portion of a file specification is not supplied. The filename extension defaults, for example, depend on the kind of operation being performed.

The device name, if omitted, is normally assumed to be the system device, and most systems also allow the user to omit the unit number. If omitted, the unit number is assumed to be unit number 0.

In addition to relying on defaults in the file specification, the user can also put an asterisk in place of a file name, file name extension, account number, or version number to indicate a class of files. The asterisk convention, also called the *wildcard convention*, is commonly used in PDP-11 operating systems when performing the same operation on related files. For example, the file specification DP1:[2,1]PROG.* refers to all files on DP1: under account [2,1] with a file name PROG and any extension. The file specification DK:[*,*]FILE.SAV refers to the files under all accounts on drive unit 0 named FILE.SAV.

▪ User Interfaces Let You Communicate with Your System

User interface refers to both the software that passes information between a user and a system and the language that a system and a user use to communicate. In the latter sense, a user interface consists of commands and messages. Commands are the instructions that the user types on a terminal keyboard (or gives to a batch processor) to tell the system what to do. Messages are the text that a system prints on a terminal that tells the user what is going on; for example, prompting messages, announcements, and error messages. This section discusses commands — the portion of the user interface that tells the system what to do, and prompting messages — the messages the system prints when it is ready to receive commands or information.

There are basically four types of commands used in PDP-11 operating systems:

- Special terminal commands which use keys on a terminal for special functions.
- Monitor or command language commands — used to request services from the system as a whole.
- I/O commands — used to direct any kind of I/O operation (often a part of monitor commands).
- System program commands — used in system programs that perform operations relevant only to the individual program.

Since system program commands are relevant only for individual system programs, and not for operating systems in general, this section discusses only monitor and command language commands, I/O commands, and special terminal commands.

Special Terminal Commands

Special terminal commands involve a set of keys or key combinations that, when typed on a terminal, perform special functions. For example, a user normally types the carriage return key at the end of an input command string to send the command to the system, which responds immediately by performing a carriage return and linefeed on the terminal. The key labeled RUBOUT or DELETE is used to delete the last character typed on the input line.

The most significant special terminal commands are those used with the key labeled CTRL (control). When the CTRL key is held down (like the shift key) and another key is typed, a control character is sent to the system to indicate that an operation is to be performed.

For example, a line currently being entered (whether as part of a command or as text) will be ignored by the system if you type a CTRL/U combination. You can then enter a new input line. The CTRL/U function is the same as typing successive RUBOUT keys to the beginning of a line. CTRL/U is standard on PDP-11 operating systems.

Another example is the CTRL/O function. If, during the printing of a long message or a listing on the terminal, you decide you are printing the wrong file, you can type a CTRL/O to stop the terminal output. If you wish to stop and then resume output to a terminal, you type CTRL/S to stop it and then CTRL/Q to resume. CTRL/O, CTRL/S, and CTRL/Q are standard functions on PDP-11 operating systems.

▪ Monitor and Command Language Commands

The primary system/user interface is provided in PDP-11 operating systems by either monitor software or special command language interface programs that run under the monitor. The monitor software and command languages allow the user to request the system to set system parameters, load and run programs, and control program execution.

An input command line consists of the command name (an English word that describes the operation to be performed) followed by a space and a command argument. The command to run a program is the word RUN followed by the name of the file containing the program, for example. If the command name is long, it can usually be abbreviated.

In the RT-11 system, a monitor component called the keyboard monitor performs the function of notifying the user that the monitor is ready for input by printing a period at the left margin. The user enters a command string on the same line and terminates the command string by typing the carriage return key.

In the RSTS/E system, there are four keyboard monitors that share the responsibility for interpreting commands — DCL, BASIC-PLUS, RSX, and RT-11. All of these interpret sets of system commands, that is, words followed by optional command parameters. These system commands allow users to perform all the fundamental functions required to use the RSTS/E system, such as logging on and off, and running programs.

▪ I/O Commands

As mentioned above, users communicate their intentions to process data files by issuing I/O commands consisting of at least one file specification. Normally, the I/O commands used in a system are standard throughout that system; in addition, most PDP-11 operating systems share the same basic I/O command string format.

Three command string formats are generally available — Digital Command Language (DCL), the Concise Command Language (CCL), and the older, less convenient Command String Interpreter (CSI) formats. Under DCL and CCL, the command, input, and output file specifications and options may all be entered in a single line in response to the system prompt.

- COPY MYFIL YOURFIL

Omitted information will be prompted for by the system until the command is complete in its general format:

- COMMAND input filespec output filespec/option

Because DCL and CCL use English-like commands and options, they are easy to learn and use.

The older CSI requires several more steps — in response to the system prompt, the user enters a RUN command and the name of a program to be run, e.g., PIP (Peripheral Interchange Program). The response is a command level prompt for file specifications:

- RUN PIP

  * YOURFIL = MYFIL.

The general format, including single-letter switches, is

- RUN Program

  * output filespec = input filespec/switch

To return to the monitor level, the user types Control C.

Command string switches are simply ways of appending qualifying information to an I/O command string. The switches used vary from program to program. They are not usually required in an I/O command string, because most programs assume default values for any switch.

Digital Command Language is a quick-to-learn command language that can be used by both interactive and batch-processed jobs for interactive program development, device and data file manipulation, and program execution and control.

Commands are composed of English words; command parameters such as file name specification and options can follow the command on the same line, or can be printed on subsequent lines in response to the system prompt.

In order to make DCL friendlier, Digital has supplied it with extensive facilities that both guide the user on the proper operation of the commands and supply explanations of system messages. In addition, through the use of defaults, DCL relieves users of many routine decisions and much redundant typing in order to complete parameters and options. Of course, the users can override the defaults in any command by using simple command options. Abbreviations also speed up the command typing procedure — users can type the shortest unique form of both commands and parameters. File specifications for DCL can be as simple as the name of the file only, or as detailed as a full listing of network, node device (including type, controller, and unit), directory, file name, file type, and version number.

Though there are more than a hundred DCL commands, users can also program and store commands of their own, and then use them just as the Digital-supplied commands are used.

Digital Command Language is a company standard that makes movement from one system to another easier by providing consistent formats and syntax. It is available on RT-11, MicroRSX, RSX-11M, RSX-11M-PLUS, MicroRSTS, RSTS/E, and VAX/VMS operating systems.

In the RSX-11 systems, an additional command interface called the *Monitor Console Routine* (MCR) allows the user to perform system level operations. There are two kinds of commands that MCR accepts—general user commands and privileged user commands. General user commands provide system information, run programs, and mount and dismount devices. Privileged user commands control system operation and set system parameters.

▪ Programmed Requests Provide Access to System Services

All PDP-11 operating systems provide access to their numerous services through requests that programs or tasks can issue during execution. A programmed request inserted directly into the program provides the mechanism.

Under the RT-11 system, MACRO-11 programmers may use programmed requests to perform file manipulation, data transfer, and such other system services as loading device handlers, setting a mark time for asynchronous routines, suspending a program, and calling the Command String Interpreter (CSI).

In the RSTS/E system, users have access to the monitor's services through system function calls. The function calls allow a program to control terminal operation, to read and write core common strings, and to issue calls, in turn, to the system file processor. File processor calls, in turn, enable a program to set program run priority and privileges, scan a file specification, assign devices, set terminal characteristics, and perform directory operations. When the function operation is performed, the program continues execution.

The RSX-11 executive includes programmed services called executive directives. Directives can be executed in MACRO programs using system macro calls provided with the system. The directives allow a program to obtain system information, to control task execution, to declare significant events, and to perform I/O operations. The RSX-11M operating system includes programmed file control services that enable the programmer to perform record-oriented and block-oriented I/O operations.

▪ System Utilities Perform Useful System-level Operations

PDP-11 operating systems provide, in general, three kinds of system *utility* programs: program development utilities, file management utilities, and special system management utilities.

Most *system management utilities* included in an operating system depend on the function the operating system serves. For example, RSX-11M, RT-11, and RSTS/E include system error logging and report programs. RSTS/E and RSX-11M-PLUS include user accounting programs. The chapters on specific operating systems will give you an idea of some of the system management utilities associated with each system. For more information refer to Chapter 17, "File Management Utilities."

Chapter 3 ▪ The RSX Family

RSX Offers Upward *and* Downward Compatibility in a Realtime, Multitasking Environment

The four-member RSX family of operating systems offers an unprecedented combination of power, flexibility, and performance to support both multiuser realtime applications and multiuser program development facilities.

RSX operating systems are designed to use little memory and to place minimal load on the CPU so that more memory and processing power are available for application programs. The systems can be generated to run in a variety of hardware and application environments—from small dedicated laboratory or industrial control systems to large multiuser information management systems.

The RSX systems run on PDP-11 microcomputers and minicomputers—a range of compatible processors with a common set of instructions and architecture. Because of these common characteristics, the processors are upward compatible. This means you can increase the size of the processor as your computing needs grow and move your RSX operating system and programs to the new processor.

Because RSX systems support a broad range of programming languages including FORTRAN IV, FORTRAN-77, BASIC-PLUS-2, COBOL-81, DIBOL-83, and PASCAL/RSX, you can choose the ones most suited to your application. A variety of software is also available to connect your RSX system with other systems. Comprehensive data management services are available to help you to manipulate files and data to suit your application.

Each RSX operating system has unique performance features that make it optimal for a specific range of PDP-11s. Through the selection of options and hardware, the RSX software can be tailored to meet the specific requirements of the application. This allows you to select the optimum size to match your needs.

The RSX Family Offers a Choice of Systems

RSX realtime operating systems provide a reliable, high-performance environment for rapid response to realtime demands as well as to less time-critical activities such as program development. The RSX family comprises four compatible realtime multiprogramming operating systems—RSX-11M, a compact, efficient operating system; RSX-11M-PLUS, a high-performance superset of RSX-11M; Micro/RSX, an extended subset of RSX-11M-PLUS designed for the Micro/PDP-11; and RSX-11S, a small execute-only version of RSX-11M for dedicated application environments.

RSX-11M-PLUS, RSX-11M, and RSX-11S are the industry's leading multiuser realtime operating systems.

- Micro/RSX

  Micro/RSX is an extended subset of the multiuser, multitasking RSX-11M-PLUS operating system. As the newest member of the RSX family, Micro/RSX was designed for use with the MicroPDP-11 and is a customer-installable, easy-to-use system in both realtime and timesharing environments.

  Micro/RSX is offered in two packages. The *Base Kit* provides the full RSX-11M-PLUS executive, appropriate utilities and device drivers, support for user-mode program development in high-level languages, and a user documentation kit. The *Advanced Programmer's Kit* is an add-on to the Base Kit and includes the software and documentation necessary for MACRO-11 and privileged program development. This includes a MACRO assembler, a librarian, an online debugging tool, and system libraries specifically designed to support privileged programming. The Advanced Programmer's Kit also includes the *data terminal emulator* and *file transfer utility* that allow for easy file transfer between a Micro/RSX system and any other RSX, VMS, or P/OS system. Communications between any of these systems are established through a terminal line.

  Micro/RSX also has Professional 300 series diskette-exchange capability. With the use of the Professional Tool Kit, programs can be developed for use on the Professional 300 systems. Like RSX-11M-PLUS, Micro/RSX also provides a migration path directly to Digital's 32-bit VAX/VMS operating system environment.

- RSX-11M-PLUS

  RSX-11M-PLUS provides the optimal multiuser system software for Digital's newest processors in the PDP-11 family. As the superset member of the RSX family of operating systems, RSX-11M-PLUS offers all of this family's capabilities.

  RSX-11M-PLUS takes advantage of the expanded addressing capabilities of some of Digital's newest PDP-11 processors while retaining the superior reliability and the successful architecture of RSX-11M. RSX-11M-PLUS uses hardware features in these PDP-11 processors that are not available in other members of the PDP-11 family. With the use of supervisor-mode library routines and separate user-mode instruction and data space, an RSX-11M-PLUS task can address up to 196 Kbytes of memory. In addition, RSX-11M-PLUS supports multistream batch, system accounting, dynamic dual-ported disks, additional memory management capabilities, and more simultaneous tasks and terminals than RSX-11M.

▪ RSX-11M

The RSX-11M operating system is the original member of the RSX family. It offers a large portion of the capabilities contained in RSX-11M-PLUS. RSX-11M excels in performance on small- and medium-size PDP-11 systems. It is designed to support factory automation, laboratory data acquisition and control, graphics, process monitoring, process control, communications, and other applications that demand immediate response. Its multiprogramming capabilities permit realtime activities to execute concurrently with such activities as program development, text editing, and data management.

▪ RSX-11S

RSX-11S is a memory-resident subset of RSX-11M. As a result, a file system is not supported on RSX-11S. RSX-11S is used as an extremely efficient execute-only system. RSX-11S is generally used in environments, such as the floor of a manufacturing plant, in which a disk cannot safely operate.

RSX-11S provides excellent online process control. Because all programs are memory-resident, response is extremely fast. Tasks for an RSX-11S system are developed on computers running the RSX-11M, RSX-11M-PLUS, or VAX/VMS operating system. Such tasks are then loaded into the RSX-11S system image by using a supplied host utility, or the RSX-11S Online Task Loader (OTL), or by downline loading if both the host and the RSX-11S system have DECnet or DECdataway support.

▪ Markets Suited to RSX Family Operating Systems

The RSX family of operating systems offers a strong base for multiuser realtime activities as well as for multiuser program development and general purpose application program execution. As such, RSX operating systems are well suited to a wide variety of markets and applications.

The realtime technical market is the place in which Digital's RSX operating systems have traditionally excelled. RSX operating systems are used in industries requiring extremely responsive multitasking support such as industrial automation, laboratory and industrial data acquisition and control, communications, robotics, and automatic testing. RSX systems are also proving essential in commercial markets with growing realtime needs, such as in airline reservation systems and electronic funds transfer systems.

RSX operating systems also provide tools to support general purpose application development and maintenance. This includes a comprehensive selection of application tools designed to shorten the program development cycle, including support for numerous programming languages, compact interactive editors, online debuggers, trace facilities, and dump analyzers. These facilities provide powerful tools for Original Equipment Manufacturers (OEMs) and others to develop and maintain applications targeted for specific markets.

RSX systems are typically thought of as complex systems for the technical user. However, with the newest member of the family, Micro/RSX, even the least experienced computer users can use RSX systems to solve their business problems. Micro/RSX is a very user-friendly version of RSX-11M-PLUS and is intended for small teams of users running on MicroPDP-11 processors. Although geared for commercial use (e.g., law or real estate office), many technical users (e.g., medical and scientific labs) are finding Micro/RSX best suits their needs for a customer-installable version of the powerful RSX-11M-PLUS realtime, multitasking, multiuser operating system.

▪ Strengths and Benefits of RSX Systems

All the RSX-11 systems are designed to meet realtime demands efficiently and effectively. RSX-11 systems feature

- Multiuser and Multitasking Support.
- Realtime Response.
- Interactive Program Development and Execution.
- Extensive Networking Support.
- Wide Range of Programming Languages.
- Wide Range of User and System Utilities.
- User or Application Environment Tailoring.
- Low Overhead.
- Customer-Installable Version for Computer-Novice User.
- Mature and Proven Software with over 50,000 Installations.
- Compatibility Up to VAX/VMS and Down to Professional Series P/OS.

Table 3-1 ▪ Comparison Chart of RSX Systems

	Micro/RSX	RSX-11M-PLUS	RSX-11M	RSX-11S
Realtime multiprogramming	X	X	X	X
Multiuser task support	X	X	X	
Event-driven priority scheduling	X	X	X	X
250 task priority levels	X	X	X	X
Dynamic memory allocation	X	X	X	
Task checkpointing to disk	X	X	X	X
Automatic memory compaction	X	X	X	
System-controlled partition	X	X	X	X
Memory Management Unit support	X	X	X	X
Extensive file-processing, file-sharing, data protection, and data management facilities	X	X	X	
An indirect command file processor	X	X	X	
Intertask communication and task sharing	X	X	X	X

	Micro/RSX	RSX-11M-PLUS	RSX-11M	RSX-11S
The easy-to-use DCL—Digital Command Language—interface (compatible with that used on the VAX/VMS operating system)	X	X	X	
EDT, Digital's easy-to-use, flexible, screen editor	X	X	X	
A comprehensive selection of high-level languages	X	X	X	X
System library routines	X	X	X	X
Clustered libraries support	X	X	X	
Online I/O exerciser	X	X	X	
Error-logging utilities	X	X	X	
A comprehensive selection of program development tools and utilities	X	X	X	X
Utilities for system management and maintenance	X	X	X	
Help files	X	X	X	

(cont.)

Table 3-1 ▪ Comparison Chart of RSX Systems (Cont.)

	Micro/RSX	RSX-11M-PLUS	RSX-11M	RSX-11S
Lineprinter spooling	X	X	X	
DECnet support	X	X	X	X
Internet capabilities	X	X	X	
Ethernet support	X	X	X	X
Packetnet system interfaces		X	X	
Remote command terminal support	X	X	X	X
Overlapped disk seeks	X	X		
Disk-seek optimization routines	X	X		
Rotational disk latency optimization	X	X		
Shared disks via dynamic dual pathing		X		
Shadow Recording support		X		
Complete batch-spooling and transparent-spooling utilities	X	X		
Virtual terminal support	X	X		

	Micro/RSX	RSX-11M-PLUS	RSX-11M	RSX-11S
Increased Dynamic Storage Region (pool)	X	X		
User-mode Instruction and Data (I- and D-) space support	X	X		
The ability to map shared libraries in supervisor mode	X	X		
Accounting	X	X		
Disk Data Caching	X	X		
Support for Ethernet Terminal Servers using Local Area Transport protocol	X	X		
Support for Digital multinational character set	X	X		
Logical names for file specifications	X	X		
Named directories	X	X		
Pregenerated versions available	X	X		
Password encryption	X	X		
Customer installable, interactive procedure	X			

(cont.)

Table 3-1 ▪ Comparison Chart of RSX Systems (Cont.)

	Micro/RSX	RSX-11M-PLUS	RSX-11M	RSX-11S
Tutorial documentation	X			
Menu-driven backup and restore utility	X			
Customer installable layered products	X			
A memory-resident executive				X
A dedicated multitask environment				X
Support for memory-resident overlaid tasks	X	X	X	X
System Image Preservation (SIP) program				X
The Basic MCR (Monitor Console Routine) user interface, a subset of the RSX-11M MCR Console Routine				X

▪ Operating System Features

In addition to controlling applications programs, RSX operating systems provide a wealth of utilities and routines that cover

- System Management and Maintenance.
- System Generation and Shutdown.
- Pregenerated Systems.
- User Environment Testing.
- Virtual Monitor Console Routine.
- The SHUTUP Program.
- User Authorization.
- User Identification Codes.
- Account File Maintenance.
- Controlling System Resources.

System Management and Maintenance

System-management personnel are responsible for overall system control, operation, and maintenance. RSX operating systems support a wealth of programs to test, monitor, maintain, and customize RSX-11 operating systems during and after system generation.

These programs let system managers create and maintain a list of valid users and User Identification Codes. System managers can also broadcast messages to all terminals and stop the system after issuing timed warning messages.

The indirect command-file processor allows all regularly scheduled maintenance activities to be stored as automated command sequences. Using the automatic scheduling feature of the RUN command, these procedures can be initiated automatically at specified intervals.

An operator uses a command language to control operations, check system status, and run utility programs. The system manager often designates one or more users to perform operator functions. A system does not require an operator, but it can have one or several persons performing operation functions that include

- System startup and shutdown.
- Task control (changing task priorities and killing tasks, for example).
- Device allocation.
- Volume mount and dismount request servicing.
- Online disk and magnetic-tape volume and file backup.
- Software maintenance update installation.
- Diagnostic execution.

System Generation and Shutdown

With the RSX system generation utility, SYSGEN, an operator can define the specific hardware configuration and select software options. Through an interactive question-and-answer session, the operator can develop a system tailored to the application's requirements. With the answers to many of SYSGEN's questions saved in a file, an operator can use that saved answer file later to modify the system or to generate another RSX system with the same capabilities.

SYSGEN contains two options that can ease the operator's involvement in the system-generation procedure and minimize the number of questions to be answered. The Autoconfigure program probes the hardware configuration and, in most cases, provides SYSGEN with a complete and accurate hardware configuration. This can eliminate the need to answer most or all of the SYSGEN questions concerning peripheral devices. On RSX-11M, the Standard Function System option and, on RSX-11M-PLUS, the full-functionality option automatically produce a mapped operating system that has most operating system capabilities. If these options are used, the number of SYSGEN questions is reduced dramatically.

Pregenerated Systems

Digital offers two RSX-supported, pregenerated systems for customers who do not need special environment tailoring. One system is the *RSX-11M-PLUS RL02 Kit* and the other is the *Micro/RSX Kit*.

The first is a pregenerated subset of RSX-11M-PLUS and is distributed on RL02 disk. Most, but not all, RSX-11M-PLUS features and device support (including error logging) are provided on the kit. Loadable driver support is also included so that you can write your own device drivers. Support for Shadow Recording and Console Logging is provided only on Instruction and Data space systems. Executive and privileged task sources *are not* included in the RL02 distribution.

The second kit is the Micro/RSX Kit. Micro/RSX is a pregenerated system and, as such, has neither the SYSGEN requirement nor the ability to be system generated. Its interactive installation procedure lets you automatically install the operating system and help files. This makes Micro/RSX easy to use and contributes to its low system management overhead.

For configuration details on both systems, refer to the RSX-11M-PLUS and/or Micro/RSX Software Product Description (SDP).

User Environment Test Package

On RSX-11M-PLUS systems, once the SYSGEN process is complete, the User Environment Test Package (UETP) verifies the integrity and operation of the newly generated system. UETP consists of several indirect command files that verify the presence and operation of devices, test the basic executive features, and verify the presence of system utilities.

UETP consists of five test modules—Load Test, I/O Exerciser Test, Utilities Test, MCR Command Test, and Interactive Utilities Test. The user can select the test to run, indicate how many times to run UETP, select or omit extended comments at the start of each test, and exclude specific devices from testing.

Virtual Monitor Console Routine

The Virtual Monitor Console Routine (VMR) is a system program that allows complete interactive configuration of an RSX-11 system.

Containing a subset of the Monitor Console Routine (MCR) commands, VMR is used to make the same changes to the system image file on a disk that can be made to the running system with MCR. Some of these changes include setting the size of pool, creating partitions, loading and unloading drivers, and installing and fixing tasks. The advantage of using VMR is that a system image file can be almost completely configured online before it is booted.

The use of VMR on RSX-11M-PLUS, RSX-11M, and VAX/VMS systems to generate an RSX-11S system is discussed later in this chapter.

The SHUTUP Program

With the SHUTUP program, a privileged user can shut down an RSX-11 system in an orderly fashion. SHUTUP prompts for the number of minutes to wait before shutdown, the number of minutes between shutdown messages, and the number of minutes to wait before disabling logins.

Before halting the system, SHUTUP performs such cleanup functions as logging off all logged-in terminals; submitting a user-written SHUTUP indirect command file for execution; stopping, if present, the Queue Manager, Console Logger, and Error Logger; deallocating checkpoint space; and dismounting devices.

User Authorization

Multiuser protection, a system-generation option, allows RSX-11M-PLUS or RSX-11M installations to monitor and control individual users of the system. This option protects against destructive interference among users.

Use of an RSX-11M-PLUS, RSX-11M, or Micro/RSX system is controlled by setting up accounts. All three systems provide an Account File Maintenance Program for creating and maintaining a multiuser account file.

▪ USER IDENTIFICATION CODES

System management assigns each RSX-11M-PLUS, RSX-11M, or Micro/RSX user a two-number identification code. Called a User Identification Code (UIC), it is enclosed in brackets and used (with password) for logging in. The number is in the form of <g,m>, "g" giving the user's group number, and "m" giving the user's member number. Under RSX's default file protection setup, users with the same group number can use each other's files without hindrance.

The group number also determines whether the user is privileged or nonprivileged: a group number of 10 or less signifies a privileged user. Installations usually have only a few privileged users. The system manager and the operators are always privileged. Privileged users have access to every part of the operating system. Nonprivileged users can use most of the operating system, but they cannot change it. Nonprivileged users can issue a DCL SHOW TIME command, for example, and the system will respond with the currently set date and time. Privileged users can issue a SET TIME command to change the system time.

Whenever a user tries to log onto a system, the system checks the account file and determines whether or not the user should be allowed access to the system. The account file describes all the UICs that have been authorized for use and their privileges. One UIC can have several users, each with an individual password.

Account File Maintenance Program

The Account File Maintenance Program (ACNT) is an interactive program that allows a privileged user — usually the system manager or an operator — to

- Create the account file.
- Add new accounts to the file.
- Examine individual account entries.
- Modify individual account entries.
- List all the account entries in the file.
- Delete an account from the file.

When activated, ACNT lists the options and requests the user to select one. According to the option selected, it responds by requesting further input or by displaying information.

Each account entry includes the following information:

- The UIC.
- The password.
- The user's system device.
- The first and last name.
- The date and time of user's most recent login.
- The number of times the user has logged into the system.
- The default command language interface (CLI).

Nonprivileged users can run the ACNT program to change the account entry descriptions or to change their own passwords.

RSX-11M-PLUS and Micro/RSX account entries also include a session identifier and a user account number. Both systems maintain a record of CPU time per user, connect time, and number of pages printed. This information can be processed through user-supplied routines like a billing program.

Data privacy and system security are based on the UIC and assigned by system management, volume protection codes, and file protection codes.

- MONITORING SYSTEM USE

  RSX-11M-PLUS, RSX-11M, and Micro/RSX include tools that monitor how a system is being used. System managers and users can then use this information to take best advantage of the system's resources.

- RESOURCE MONITORING DISPLAY

  The Resource Monitoring Display (RMD) provides information about the active tasks in the operating system and the availability of system resources. This information includes the active tasks, their location in memory, the amount of memory they occupy, and available pool space. RMD generates dynamic displays on videoterminals and "snapshot" displays on hardcopy terminals.

  The information is presented in easily comprehensible graphic form. RMD can also help locate certain system lockout problems or bugs in an application program and/or system-level software.

- SOFTWARE PERFORMANCE MONITORS

  Software Performance Monitors (SPM), including SPM-11M-PLUS and SPM-11M, are high-resolution, low-overhead, event-driven performance monitors. Optionally available to run under RSX-11M-PLUS and RSX-11M, these monitors allow users to access the impact of individual tasks on system resources and determine which tasks use the most resources and which tasks wait the longest for resources. Users can also identify resources used heavily by the total system workload to aid in locating bottlenecks. A flexible set of controls for data collection allows the user to measure and generate reports on usage and waiting times for the CPU, memory, I/O devices, the file system, and the task loader.

- RESOURCE ACCOUNTING

  On RSX-11M-PLUS and Micro/RSX systems, Resource Accounting, a system generation option, provides a transaction file of system usage information. Accounting gathers data for both the user and the system. With this data, system management can bill individual users for the used resources and can measure overall system usage.

  Resource Accounting gathers and saves in the transaction file information about the following topics: users; tasks; system; logons and invalid logons; device allocation, deallocation, mount, and dismount; print jobs; system time changes; and device usage.

- CONSOLE LOGGER

  The Console Logger consists of a driver and the Console Output Task (COT) that handle I/O to the console output device and record time-stamped system messages on a terminal or in a log file or both. Support for console logging is a SYSGEN option.

The COT handles messages sent to the console device. It allows the system manager to forward messages to an alternative terminal or to a file. The system manager can also forward messages to the console device or logging device. The Console Driver sends all messages to COT, and COT forwards them to the selected terminal or logging device. Both COT and the Console Driver are supported on all RSX-11M-PLUS systems and on mapped RSX-11M systems.

Maintaining Volumes

With RSX's many volume maintenance facilities, users can back up files onto disks and tapes, locate bad blocks on the volumes, consolidate disk data-storage areas, and verify the contents of the volumes. In addition, users on RSX-11M-PLUS systems can back up all information as it is being written to a Files-11 disk by using Shadow Recording.

- DISK VOLUME FORMATTER

  The Disk Volume Formatter (FMT) formats and verifies disk cartridge, disk pack, fixed-media disk, and flexible-disk volumes under an RSX-11M-PLUS or RSX-11M operating system that includes online formatting support, which is a SYSGEN option.

  The disks can be completely formatted in normal operating mode or formatted on an individual sector basis in manual operating mode.

  In general, FMT performs the following:

  - Writes a complete header for each section of the disk it is formatting.
  - Verifies the address contents of each sector header.
  - Sets the density for Digital's RX02 floppy diskettes.
  - Lets the user specify an error limit for the volume being formatted (FMT terminates processing if the error limit is reached).
  - Lets the Bad Block Locator utility be run (spawned) if the system permits spawned tasks.

- BAD BLOCK LOCATOR

  The Bad Block Locator (BAD) utility tests disks and magnetic tapes for the location and number of bad blocks and records this bad-block information on the volume. The MCR INI (initialize volume) command is then used to allocate the bad blocks to a specific file. The bad blocks are marked as "in use" and thus cannot be allocated to other files.

BAD supports any last-track device, as well as vendor-supplied cartridges that do not have a prerecorded manufacturer's bad-sector file on the last track. Users can use BAD in its task version, which runs at the same time as other tasks, or in its stand-alone version, which runs by itself on the computer. The stand-alone version must be used if the system has only one disk drive.

- BAD BLOCK REPLACEMENT CONTROL TASK

  The Bad Block Replacement Control Task (RCT) handles bad-block replacement and recovery on Mass Storage Control Protocol (MSCP) disks like the RA80 or RA81. Bad-block handling on MSCP disks consists of four stages — detection, notification, replacement, and revectoring.

  The disk controller detects bad blocks and notifies the driver. The driver activates RCT. RCT performs all the bad-block replacement functions that enable the controller to revector (redirect) I/O from the bad block to the replacement block.

  RCT performs the following bad-block replacement functions:

  - Stores data from the bad block.
  - Allocates a replacement block.
  - Updates data structures on the disk.
  - Initializes the replacement block.

  RCT also performs replacement and recovery on MSCP disks that went offline during bad-block replacement or before the contents of a write-back cache were copied to the disk. If RCT determines that bad-block replacement was partially completed when the disk went offline, RCT completes the bad-block replacement process. If RCT determines that the write-back cache was not copied to the disk before the disk went offline, RCT software write-locks the disk so that the contents of the write-back cache are preserved.

- BACKUP AND RESTORE UTILITY

  With the Backup and Restore Utility (BRU), users can back up and restore Files-11 volumes. BRU transfers files from a volume to a backup volume (or volumes) to ensure that a copy of the files is available in case the original files are destroyed. If the original files are destroyed, or if for any other reason the copy needs to be retrieved, users can restore the backup files with BRU commands. Users can run BRU either at the same time as other tasks or stand-alone.

Backup and restore operations that take place on disk and tape volumes are disk to tape — for backup operations and tape to disk — for either backup or restore operations. In addition to these basic data transfer functions, BRU provides command qualifiers to

- Initialize disks.
- Perform selective backup and restore operations.
- Control such tape processing as density, length, ANSI tape labeling, rewinding, and appending.
- Perform volume and data checking.
- Display such information as backup set names and file names.

BRU reallocates and consolidates the disk storage area. It concatenates files and their extensions into contiguous blocks whenever possible, and it can reduce the number of retrieval pointers and file headers required for the same files on the new disk volume.

A BRU operation begins with data on one disk and ends with the same data on another disk, in compressed form.

- DISK SAVE AND COMPRESS UTILITY

The Disk Save and Compress (DSC) utility copies a Files-11 structured disk either to disk or to tape from a DSC-created tape. At the same time, DSC reallocates and consolidates the disk data storage area. It concatenates files and their extensions into contiguous blocks whenever possible and, therefore, reduces the number of retrieval pointers and file headers required for the same files on the new volume.

DSC copies files that are scattered randomly over a disk volume to a new volume, without the intervening spaces. This eliminates unused space between files and also reduces the time required to access them.

After a DSC copy operation, individual files are written in available contiguous blocks, and the blocks available for new files are located in a contiguous area at the end of the new volume. If the contents of one disk are transferred to a disk with a larger capacity, the new disk takes on the attributes of the original disk, except that additional storage space is available.

- FILE STRUCTURE VERIFICATION UTILITY

  For Files-11 volumes, the File Structure Verification (VFY) utility can perform the following functions:

  - Check the readability and validity of a file-structured volume (default function).
  - Print the number of available blocks on a file-structured volume.
  - Search for files in the index file that are not in any directory (that is, files that for some reason cannot be accessed by filename).
  - Validate directories against the files they list.
  - List all files in the index file, showing the file ID, filename, and owner.
  - Mark as "used" any blocks that appear to be available, but are actually allocated to a file.
  - Rebuild the storage allocation map so that it properly reflects the information in the index file.
  - Restore files that are marked for deletion.
  - Delete bad file headers.
  - Perform a read check on every allocated block on a file-structured volume.

Controlling System Resources

On RSX-11M-PLUS, RSX-11M, and Micro/RSX systems, the Queue Manager (QMG) provides for the orderly processing of print jobs and, on RSX-11M-PLUS, batch jobs. Most users use the Queue Manager without being aware of its presence. All requests to print a listing, submit a batch job, and create a spooled listing or map from a system program are passed automatically to the Queue Manager, which places them in the appropriate queues.

QMG includes privileged commands used by the system manager to set up the QMG for the installation. Depending on the needs of the installation, the manager can choose the number of queues to be set up and can choose where the output of these queues will be directed. A system can have as many as 16 output devices to which QMG directs output. The manager can also specify that certain kinds of print jobs will go to one or another lineprinter.

Some installations may have user-written output processors that pass jobs to devices other than lineprinters. QMG can pass jobs to most kinds of devices. These include queues for electrostatic plotters, terminals and magnetic tapes. QMG can direct output to any record-oriented output device. It also can direct output to application tasks or networks.

When a lineprinter or other output device is controlled by the Queue Manager, it is said to be a spooled device. Spooled devices are initialized by the system manager with certain characteristics. For example, a lineprinter can be initialized to print both uppercase and lowercase characters on a previously defined special form. Using PRINT command qualifiers, operators can specify the characteristics they want, such as printing with uppercase characters only. An operator, and in some instances any user, can control a job queue by changing job priorities, holding a job, or canceling a job.

- PRINTING FILES
The Queue Manager handles the orderly printing of files for an RSX-11M-PLUS, RSX-11M, or Micro/RSX system through software tasks called despoolers, or print processors. There is a print processor for every lineprinter on a system. Even if a system does not have a hardware device of the lineprinter type, it will have some device designated as the system output device that takes the place of the lineprinter. This output includes system task requests for maps or listings, logs from batch jobs, and print jobs entered through the PRINT command, as well as output from applications tasks unique to an installation.

  A print job can specify the forms required, the number of copies, the print priority, holding an entry, deleting files after printing, and printing after a specified time.

  The print job can contain one or more files to be printed. Print jobs can be submitted by an interactive user, a program, and on RSX-11M-PLUS and Micro/RSX, by a batch job. Print jobs are automatically submitted at the end of a batch job.

  Each print job has a large, easily read job header and file header burst pages to identify print requests and files within a print request. Both types of headers contain identification and general accounting information.

- BATCH PROCESSING
RSX-11M-PLUS and Micro/RSX have a multistream batch processing capability. The operations personnel can control the number of batch streams that can run.

  Batch jobs can be submitted by an interactive user, a program, or another batch job. When the number of batch jobs submitted exceeds the number of streams, the remainder of the batch jobs are held in a batch input queue. As with the spool queues, the operator can control the batch job queue by changing job priority, holding a job, or killing a job.

  Volume mount commands issued in a batch job can request a generic device, such as a disk, or specific device unit, such as disk-drive unit 2. The batch job waits until the operator satisfies the mount request, while other batch jobs proceed.

▪ User Interfaces

A *user interface* consists of commands and messages. Commands are the instructions that the user types on a terminal keyboard (or gives to a batch processor) to tell the system what to do. Messages are the text that a system prints on a terminal that tells the user what is going on; for example, prompting messages, announcements, and error messages.

RSX users usually interact with applications via an online terminal. To aid in the development of interactive and realtime applications, RSX users can write, compile, taskbuild, and test programs interactively. RSX users can also design applications that require a high degree of data sharing, intertask communication, and file manipulation. Users can directly control these operations through the operating system command languages.

System Command Languages

RSX-11M-PLUS and RSX-11M users determine which of one or more command line interpreters (CLIs) are installed. DCL—the Digital Command Language—and MCR—the Monitor Console Routine—are the most common. Many systems include and support other CLIs as well. Both DCL and MCR are interactive, comprehensive, and flexible command languages. Both include commands to invoke most system tasks and utilities and to set and display certain system characteristics. In general, DCL commands specify actions directly, as in the COPY and TYPE commands. MCR commands name tasks—PIP, for example—a utility used to manipulate files (such as copying and typing them).

DCL is an optional user-oriented CLI included in most systems with many users. Commands in DCL are English words that follow well-defined syntax rules. Full commands are self-documenting. DCL is designed for consistency and ease of use.

DCL is based on the command languages used on a number of Digital's operating systems. In particular, RSX-11M, RSX-11M-PLUS, and Micro/RSX DCL is designed for compatibility with VAX/VMS DCL.

DCL on RSX-11M-PLUS, RSX-11M, and Micro/RSX is a CLI task that translates DCL commands into MCR commands for execution by the system. The DCL SET DEBUG command displays on a terminal the MCR translation for any DCL command.

In general, MCR commands must be entered in exact syntax. MCR commands follow no set syntax rules, however. Most MCR commands are terse abbreviations or mnemonics.

Depending on the way they use the system and on the nature of the system itself, users may find it more convenient to use DCL or MCR, or both. All non-privileged system functions are available directly from DCL, but some privileged functions are not. All program-development facilities and all common utility functions are available from DCL.

Some DCL commands require parameters or arguments as part of the command line. If the user fails to supply a required command element, DCL supplies a prompt with one or two words indicating the general nature of the required element. DCL will also supply help information when a question mark is typed in response to a prompt.

To issue an MCR command, the user types a command string consisting of a command name and any required parameter. A parameter can be a task name, the name of a file, or a device specification. A DCL command line consists of the command name, which is a verb describing the action the system is to take, with optional qualifiers and parameters to further define the action of the command.

As a simple illustration of how DCL commands differ from MCR commands, both of the following command lines will print a copy of a file named TECSUM.TXT on the user's terminal. The first example is from a DCL terminal; the second, MCR.

```
DCL>TYPE TECSUM.TXT
MCR>PIP TI:=TECSUM.TXT
```

The system displays DCL or MCR and the angle bracket on the terminal as an explicit prompt for a command. To explain the MCR command: PIP, the Peripheral Interchange Program, is a file manipulation utility. TI: is the input pseudodevice that stands for the user's own terminal, and the equals sign is required as part of the syntax. The DCL command, on the other hand, is self-explanatory.

Because RSX-11M-PLUS and RSX-11M are designed to be tailored to the needs of each installation, not every feature of DCL and MCR is available on every system. Some commands depend on layered products that may not be available at a particular installation. Many features, particularly those on RSX-11M systems, are system-generation options that may not have been selected at the time the system was generated.

User-written CLIs

An RSX operating system installation may have special command-language requirements that neither MCR nor DCL can meet. So RSX operating systems support the capability of easily implementing a custom command language interpreter that is specific to an application. A CLI is an RSX system task and, like any other, can be written in any RSX-supported programming language. Users can write CLIs without knowledge of operating system internals. Privileged "system" code is not required. RSX systems can support multiple CLIs, with the added convenience of each terminal being preset for the desired CLI.

Indirect Command Files

To eliminate the need for typing frequently repeated sequences of commands, users can create an indirect command file — a text file that contains complete command lines or a series of commands. When the user enters the name of the indirect command file, the system processes the command lines in the file just as if they were being successively processed at the terminal.

Indirect command files also allow system queries, string substitutions, multi-level indirect command files of up to four levels, special symbol definitions, and an extensive number of directives. The directives allow users to

- Define labels.
- Define and assign values to three types of symbols — logical, numeric, and string.
- Create and access data files.
- Control the logical flow within a command file.
- Perform logical tests of internal and system states.
- Invoke subroutines.
- Determine if an invoked task exited successfully.
- Do arithmetic.
- Control time-based and parallel task execution.

There are two types of indirect command files — indirect task command files and indirect DCL/MCR command files. An indirect task command file is a sequential file containing a list of task-specific commands. Rather than typing commonly used sequences of commands, the sequence can be typed once and stored in a file. The indirect task command file is specified in place of the command line normally submitted to the task.

An indirect DCL/MCR command file contains a list of DCL or MCR commands. RSX systems have an indirect file processor for interpreting commands in a file. An indirect DCL/MCR command file can contain both normal DCL/MCR commands and special commands (known as directives) that allow the user to program the execution of the indirect command file.

Batch Jobs

In addition to executing indirect command files at a terminal, an RSX-11M-PLUS or Micro/RSX user can submit batch jobs. Batch jobs are similar to indirect command files, except that the user does not have to be present or even logged onto the system when the batch job is run.

Batch jobs execute under the control of the Queue Manager program. Because a batch job doesn't require the user to be present, jobs that take a long time to run or otherwise tie up system facilities can be run when there are fewer demands on the system—at night or on weekends, for example. Users can submit a batch job and continue using their terminals for other work.

To create a batch job, a user enters batch-specific commands, CLI commands, and data into a file. The information contained in the batch job must duplicate a complete interactive terminal session, including logging itself into the system, controlling operations, and logging itself off. Using the SUBMIT command, the batch job is then submitted to the Queue Manager for batch processing.

Tasks called batch processors pass the commands in batch jobs to the operating system. Commands to be passed by the batch processor are the same as normal system commands, except that they are preceded by a dollar sign ($). (The only exceptions are for LOGIN/HELLO and LOGOUT/BYE—these are replaced by $JOB and $EOJ, respectively.)

The batch processor uses a software terminal called a "virtual terminal" to pass commands and data to system tasks. A batch job can do almost anything from a virtual terminal that a user can do from an interactive terminal, including compiling or assembling, building tasks, and running tasks.

Users can submit batch jobs at any time the Queue Manager is active. Batch jobs can run at any time as well. System management controls the availability of batch processing.

RSX-11M-PLUS and Micro/RSX support multistream batch processing. Operations personnel can control the number of batch streams that can run. An interactive user, a program, or another batch job can submit batch jobs. When the number of batch jobs submitted exceeds the number of streams, the remainder of the batch jobs are held in a batch input queue. The operator can control the batch job queue by changing job priorities, holding a job, or canceling a job.

Complete information on the progress of batch jobs is always available. Within specified limits, a user can indicate when a job should be run (during evening hours when demands on the system are lighter, for example), and if the job should be held, released, or deleted from its queue.

HELP

Users can get information about various aspects of an RSX-11M-PLUS, RSX-11M, or Micro/RSX system through the HELP command or, for help on DCL-specific features, by typing a question mark in response to any DCL prompt.

Typing HELP on the terminal displays a list of the HELP files available. To get help on the TYPE command, for example, the user enters HELP TYPE on the terminal. In such an instance, the HELP text consists of a brief explanation of the command followed by an illustration of the syntax.

System managers can create help files that can be made available to all users to provide information on special aspects of their installations. In addition, users can create local help files for their own use.

Program Interfaces

All RSX systems are priority-scheduled and event-driven. The assigned priority and activities of the tasks determine the level of service they receive. RSX systems allow program segmentation and overlaying capabilities for programs that need more memory space than is available.

The user program environment is the task entity that the operating system schedules for execution. The following paragraphs describe how RSX systems handle task execution.

Multiprogramming

Multiprogramming, the concurrent execution of two or more tasks residing in memory, is possible because task execution almost always involves more than just CPU (Central Processor Unit) usage. A realtime task that initiates a process and then waits for the process to finish might not need CPU time while it is waiting. Therefore, while one task waits for an event to complete, the executive gives control of the CPU to another task. This happens so rapidly that many individual users seem to have control of the CPU at the same time. This illusion is often referred to as apparent concurrency.

In the RSX family, tasks are multiprogrammed by logically dividing available memory into a number of named partitions. These partitions are set up when a system is generated using the Virtual Monitor Console Routine. For maximum flexibility, privileged users can use DCL or MCR commands to establish and eliminate partitions whenever the need arises.

A partition is a contiguous area of memory with

- A name.
- A defined size.
- A fixed base address.
- A defined type.

The relationship between a task and the partition in which it runs depends on whether the partition is system- or user-controlled, and whether the system is mapped or unmapped. In general, RSX-11M and RSX-11S systems can have two kinds of partitions—system-controlled and user-controlled. Because RSX-11M-PLUS systems are larger, mapped systems, they support system-controlled partitions only.

User-controlled partitions are used when the programmer wants to handle memory allocation while executing a task. System-controlled partitions are intended when a task being executed requires the system to handle the allocation of memory.

Space in a system-controlled partition is dynamically allocated by the executive to contain as many tasks as will fit simultaneously in the partition. This allocation can involve shuffling resident tasks to arrange available space into a contiguous block large enough to contain a requested task. Only mapped systems support system-controlled partitions. Mapped and unmapped systems are discussed below.

A user-controlled partition is allocated to only one task at a time. The user has complete control over system activity in this type of partition. In RSX-11M systems, a user-controlled partition can be subdivided into as many as seven nonoverlapping subpartitions. The subpartitions occupy the identical physical memory occupied by the main partition. Tasks built to execute in the subpartitions can execute in parallel. Tasks cannot, however, be resident in a main partition and its subpartitions simultaneously. If a main partition is occupied, the subpartitions cannot be. All subpartitions can have tasks residing in them; therefore, up to seven tasks can execute in parallel within a preempted, user-controlled main partition.

The goal of subpartitioning is to reclaim large memory areas in an unmapped system. When a large task that requires a main partition is no longer active, or when it can be preempted (checkpointed), subpartitioning allows the partition space to be used for a number of smaller tasks.

Dynamic Memory Allocation

Dynamic memory allocation is an extension of the RSX multiprogrammed partition structure. Dynamic memory allocation enables the system to respond rapidly to changing requirements for system resources.

RSX systems let users load and execute more than one task in a system-controlled partition. If a loaded task does not fill the entire partition, another task can be loaded into the space above or below it, as long as the remaining contiguous physical space is large enough to contain it.

The executive keeps a record of used areas and determines free space when needed. Tasks are brought in from disk according to their priority and are loaded into the first available memory area in the partition. The executive continues to load tasks as long as sufficient contiguous physical memory is available in the partition. When a task terminates and is sent back to disk, the memory it occupied becomes available again.

Normally, a task cannot be loaded into a system-controlled partition unless sufficient contiguous space between other tasks loaded in the partition is available. When a task terminates, it can leave a space that alone is insufficient to hold another task. If combined with other unused areas, however, this space could be large enough to contain the task.

In a system-controlled partition, the noncontiguous free spaces are combined by an automatic memory-compaction feature that shuffles or moves tasks to ensure that sufficient space is available to load another task. Dynamic memory allocation is always present with RSX-11M-PLUS, although it can be turned off. It is optional on RSX-11M and RSX-11S.

Mapped and Unmapped Systems

The 16-bit architecture of the PDP-11 dictates that a task can directly address only 64 Kbytes of memory. A special hardware device, a Memory Management Unit, makes it possible to use more than 64 Kbytes of memory. The Memory Management Unit associates addresses used in tasks (virtual addresses) with actual locations in memory (physical addresses). The concepts of virtual and physical addresses are discussed later in this section.

A PDP-11 system that includes a Memory Management Unit is referred to as a mapped system. Systems without the unit are called unmapped. RSX-11M and RSX-11S systems can be mapped or unmapped. Mapped RSX-11M and RSX-11S systems can have both system-controlled and user-controlled partitions. Unmapped systems can have only user-controlled partitions.

Because all PDP-11s supported by RSX-11M-PLUS and Micro/RSX include memory management hardware, RSX-11M-PLUS and Micro/RSX systems are always mapped systems and support only system-controlled partitions.

Mapped and unmapped systems install tasks into a partition differently. In unmapped systems, a task is linked to be installed and run in a partition with a specific base address. The task cannot run at a base address different from that specified in the Task Builder command.

In mapped systems, a task can be installed in any partition large enough to contain it. The reason is that the Memory Management Unit maps the virtual addresses of a task to the actual physical addresses in which the task resides.

The memory management hardware provides automatic memory protection on RSX systems. Because each task has an absolute address range in which to execute, the memory area assigned to a task is protected from other tasks executing in the system. A task can refer to and alter memory only within the specific task area it owns.

Scheduling

Because the RSX family is designed specifically for realtime environments, event-driven scheduling is used to determine a task's eligibility to execute. The basis of event-driven task scheduling is the software priority assigned to each active task. A task's default priority is set when the task is built; it can either be changed dynamically from within a task or altered interactively via a command.

Tasks are run at a software priority level ranging from 1 (low) to 250 (high). The executive grants central processor resources to the task of highest priority that can be executed. That task retains control of the central processor until the task gets blocked. Blocking occurs when a task issues a system directive that implicitly or explicitly suspends its execution. For example, a task can issue a directive that indicates it wants to wait until an I/O operation is complete before continuing execution.

Blocking is one type of significant event. A significant event is declared whenever there is a change in system status. Whenever there is a significant event, the executive reviews the eligibility of tasks to execute.

- EVENT FLAGS

  A task can recognize that a significant event has occurred by means of event flags. In requesting a system operation (such as an I/O transfer), a task can associate an event flag when the operation is completed. When the significant event occurs, the executive sets the specified flag, then interrupts the executing task and searches for a task capable of executing. The task of highest priority that has all the resources it needs to run and that can make use of those resources will be the task that gains control of the CPU.

Event flags can be used to coordinate task execution for intertask communication. There are 96 event flags—1 through 32 are local to the task, 33 through 64 are common to all tasks, and 65 through 96 are group-global. Although group-global event flags can be used in any application, they must be used by tasks containing the group code specified when the group-global event flags were created. A task can set, clear, test, and wait for any event flag or combination of event flags to achieve efficient synchronization between itself and other tasks in the system.

For example, upon completion of I/O requests, the executive normally sets a requester-indicated event flag and declares a significant event. If a requesting task instructs the system that it cannot run until an event flag is set (signaling task I/O completion), other eligible tasks of lower priority may run.

- ROUND ROBIN SCHEDULING

Although event-driven scheduling is the primary RSX task-scheduling mechanism, it is not the only mechanism available. As an option during system generation, RSX systems allow the user to supplement event-driven task scheduling with round robin scheduling for some or all tasks.

Round robin, or first-in-first-out (FIFO), scheduling is available for a priority range specified during system generation. If there is more than one task active at a given priority level, and if round robin scheduling was included in SYSGEN, the tasks are rotated in the priority queue. A form of timesharing, round robin scheduling gives tasks of similar priority an equal share of CPU time.

- TASK CHECKPOINTING

With RSX-11M-PLUS and RSX-11M, effective multiprogramming is achieved when many tasks, residing in memory simultaneously, spend some of their residency waiting for I/O completion, waiting for synchronization with other tasks, or being unable in some way to continue execution—while one other task uses the central processor's resources.

This multiprogramming scheme normally applies only to tasks resident in memory. Once a task is in memory, the executive allows it to run until a significant event occurs, even if its memory space is required to execute a nonresident task of higher priority. However, if it is desirable to free memory for the execution of a task of higher priority, a task can be declared checkpointable when it is built, installed, or running.

A checkpointable task can be swapped out of memory when a task of higher priority requests the partition in which it is active. With checkpointing, more tasks can be brought into memory to run, thus increasing system throughput.

In RSX-11M-PLUS, RSX-11M, and Micro/RSX systems, task priority normally determines which tasks can checkpoint other tasks. A checkpointable task currently active in a partition, but of a lower priority than another task requesting the partition, can be preempted and rolled out to a disk. The task is saved on the disk exactly as it was when interrupted. When memory is available, the task of lower priority can be restored to active execution at the point where it had been interrupted.

In addition, a memory-resident task that is waiting for terminal input is always checkpointable, regardless of its priority.

- STOP-BIT SYNCHRONIZATION
RSX systems also provide a stop-bit synchronization feature that allows tasks to be stopped until an event occurs. When a task is stopped, it is blocked from further execution, its priority for memory allocation effectively drops to zero, and it can be checkpointed by any other task in the system. If checkpointed, the task remains out of memory until its stop-bit is cleared. At this time the task becomes unstopped, and its normal priority for memory allocation is restored. This feature is particularly useful when a long waiting period is anticipated.

System Directives

When a task requests the executive to perform an operation, it is called a system directive. Programmers use these directives to control both the execution and interaction of tasks. The MACRO-11 programmer usually issues directives in the form of macros defined in the system macro library. The FORTRAN programmer issues system directives in the form of calls to subroutines contained in the system object module library.

System directives affect the way the executive shares system resources among concurrently active tasks, directly or indirectly. The following list groups the directives by function into nine categories.

- Task Execution Control Directives deal principally with starting and stopping tasks. Installed tasks can be either dormant or active. The active state has three substates—ready-to-run, blocked, and stopped. Each of the Execution Control Directives (except Extend Task) results in a change of the task's state (unless the task is already in the state being requested).
- Task Status Control Directives alter the checkpointable attribute of a task and/or change the running priority of an active task.
- Informational Directives provide the issuing task with system information and parameters, such as the time of day, the task parameters, the console-switch settings, and partition or region parameters.

- Event-Associated Directives provide a means in the system for intertask and intratask synchronization and signaling.
- Trap-Associated Directives provide the user with trap facilities that allow transfer of control (software interrupts) to the executing tasks.
- I/O- and Intertask Communications-Related Directives allow tasks to access I/O devices at the driver interface level or interrupt level, to communicate with other tasks in the system, and to retrieve the command line used to start the task.
- Memory Management Directives allow a task to manipulate its virtual and logical address space and to set up and dynamically control the window-to-region mapping assignments. These directives also provide the means by which tasks can share and pass references to data and routines. Memory management concepts are discussed later in this section.
- Parent/Offspring Tasking Directives permit tasks to start other tasks, connect to tasks in order to receive status information, stop for terminal I/O, and unstop other tasks. A more detailed description of parent/offspring tasking appears later in this section.
- RSX-11M-PLUS Directives. In addition to the directives listed above, RSX-11M-PLUS includes directives that support virtual terminals, supervisor-mode library routines, variable-length send/receive data buffers, and parity error and exit AST (Asynchronous System Traps) routines.

System Traps

System traps, also called software interrupts, are another means of governing task execution. While significant events can have a systemwide scope, traps are local to a task. Traps interrupt the sequence of instruction execution in the task and cause control to be transferred to a prespecified point in the program. In this way, traps provide the ability to service certain conditions without continuously testing for their existence.

To use the system trap facility, a task must contain a trap service routine. This routine is automatically entered when the trap occurs, using the task's normal priority and privilege. If a service routine is not supplied, the action taken by the executive depends on the type of trap. There are two types of traps — Synchronous System Traps (SSTs) and Asynchronous System Traps (ASTs).

SSTs provide a means of servicing fault conditions within a task, such as memory protection violation and floating-point-unit exceptions. These conditions, which are internal to a task, occur synchronously with respect to task execution. In these cases, if an SST service routine is not included in the task, the task's execution is aborted.

ASTs are a result of a significant event occurring asynchronously with respect to a task execution. A task does not have direct or complete control over the timing of an AST's occurrence. ASTs are used for information purposes, such as signifying an I/O completion about which a task needs to know immediately.

If an AST service routine is not provided, a trap does not occur, and task execution is not interrupted.

Memory Management

Without using advanced programming techniques along with the memory management hardware available on some PDP-11s, an RSX task cannot explicitly address more than 64-Kbytes of memory. The 16-bit word size of the PDP-11 imposes this restriction on a task's addressing capability. Two programming techniques can overcome this addressing limitation—overlaying segments of a task with either disk-resident or memory-resident code and mapping to different regions of memory outside the physical limits of the current task space.

- OVERLAYING

Users can reduce the memory and/or virtual address space requirements of a task by using overlay structures created with the Overlay Description Language. RSX systems support two kinds of overlaid segments — those that reside on disk and those that reside permanently in memory.

To create an overlay, a task must first be divided into segments — a single root segment — which is always in memory — and any number of overlay segments — which are loaded into memory as required.

The overlaid segments either reside on disk and share virtual address space and physical memory with one another (disk-resident overlays), or they reside in memory and share only virtual address space with one another (memory-resident overlays).

Disk-resident overlays save space, but they introduce overhead activity because the segments must be loaded into memory each time they are needed but not present in memory. Memory-resident overlays save time. They are loaded from disk only the first time they are referred to. Thereafter, they remain in memory and are referred to by remapping. This, of course, consumes memory space.

The Task Builder enables the user to build overlaid tasks and call these overlays from disk. The use of overlaid tasks and disk-based data can significantly expand the achievable peak load of an RSX system and still maintain acceptable response time.

Several large classes of tasks can be handled effectively by an overlay structure. For example, a task that moves sequentially through a set of modules is well suited to an overlay structure. So, too, is a task that selects one of a set of modules according to the value of an item of input data.

- MEMORY MANAGEMENT DIRECTIVES

  Data is disk-based, not only because of limited memory space but also because transmission of numerous instructions or large amounts of data between tasks is practical only via disk. However, an overlaid task or a task that needs to access or transfer large amounts of data incurs a considerable amount of disk activity over and above that caused by the task's function.

  Task execution could obviously be faster if all or a greater portion of a task were resident in memory at runtime. RSX-11M-PLUS, RSX-11M, and Micro/RSX include a group of memory management directives that provide a task with this capability. These directives overcome the limited addressing restriction by allowing the task to dynamically change the physical locations that are referred to by a given range of virtual addresses. With these directives, a task can increase its execution speed by reducing its disk I/O requirements at the cost of requiring more physical memory.

  The Memory Management Unit is the PDP-11 hardware that translates (relocates) a task's 16-bit virtual address into either an 18-bit or 22-bit (depending on processor) physical address. The hardware consists of an adder, a number of registers that perform the actual address translation, and an overall internal system protection scheme.

  The basic function of memory management is to perform memory relocation and provide extended memory addressing capability for systems with greater than 64 Kbytes of physical memory. In order to perform this basic function, the memory management system uses a series of Active Page Registers (APRs). The APRs are hardware relocation registers that permit several users' programs, each starting at virtual address 0, to reside simultaneously in physical memory.

- ADDRESS SPACE

  The concepts of virtual address space, logical address space, and physical address space provide a basis for understanding the functions performed by memory management directives.

  - A task's *virtual address space* corresponds to the 64-Kbyte address range imposed by the PDP-11's 16-bit word length. The task can divide its virtual address space into segments called virtual address windows.
  - A task's *logical address space* is the total amount of physical memory to which the task has access rights. The task can divide its logical address space into various areas called regions. Each region occupies a contiguous block of memory.
  - A task's *physical address space* is the actual physical memory where the task resides and executes.

RSX-11M-PLUS, RSX-11M, and Micro/RSX supply memory management directives to assign or map a range of virtual addresses (a window) to different logical areas (regions); this enables the user to extend a task's logical address space beyond 64 Kbytes. Without these directives, a task's virtual address space and logical address space would directly correspond; thus, a single virtual address would always point to the same logical location.

- WINDOWS
To manipulate mapping virtual addresses to logical areas, the user must first divide the task's 64 Kbytes of virtual address space into segments. These segments are called virtual address windows.

When a task uses memory management directives, the executive views the relationship between the task's virtual and logical address space in terms of windows and regions. Unless a virtual address is part of an existing address window, references to that address will cause an illegal address trap to occur. Similarly, a window can be mapped only to an area that is either all or part of an existing region of the task's logical address space.

Once a task has defined the necessary windows and regions, it can issue memory-management directives to perform such operations as

- Mapping a window to all or part of a region.
- Unmapping a window from one region in order to map it to another region.
- Unmapping a window from one part of a region in order to map it to another part of the same region.

- REGIONS
A region is a portion of physical memory to which a task has access. The current window-to-region mapping context determines the part of a task's logical address space that the task can access at one time. A task's logical address space can consist of various types of regions:

- *Task region*—a contiguous block of memory in which the task runs.
- *Static common region*—an area defined at SYSGEN time or at runtime, such as a global common area.
- *Dynamic region*—a region created dynamically at runtime by issuing memory management directives.
- *Shareable regions*—on RSX-11M-PLUS only, a read-only portion of multiuser tasks

Address mapping not only extends a task's logical address space beyond 64 Kbytes but also allows the space to extend to regions that have not been linked at task-building time. One result is an increased potential for task interaction by means of shared regions. For example, a task can accommodate large amounts of data. Any number of tasks can then access that data by mapping to the region. Another result is the ability of tasks to use a greater number of common routines. Thus, tasks can map to required routines at runtime, rather than link to them at task-building time.

A region becomes part of a task's logical address space by being attached. A task can map only a region that is part of the task's logical address space. There are three ways to attach a task to a region:

- All tasks are automatically attached to regions that are linked to them at task-building time.
- A task can issue a directive to attach itself to a named static common region or a named dynamic region.
- A task can request the executive to attach another task to any region within the logical address space of the requesting task.

Attaching identifies a task as a user of a region and prevents the system from deleting a region until all user tasks have been detached from it. A task cannot indiscriminately attach to any region because each region has a protection mask to prevent unauthorized use.

Memory management directives are capable of performing a number of separate actions. The complexity of the directives requires that there be a special means of communication between the user task and the executive. This means is achieved through user data structures that allow the task to specify which directive options it wants the executive to perform and permits the executive to provide the task with details about the outcome of the action it is requesting.

▪ RSX-11M-PLUS MEMORY MANAGEMENT DIRECTIVES

RSX-11M-PLUS uses the extra set of memory mapping registers available on many of Digital's newest processors. The supervisor-mode registers can be mapped by RSX-11M-PLUS to map read-only libraries of user code, the RSX File Control Services, or other application code. This capability gives an RSX-11M-PLUS task a full 64 Kbytes of normal address space plus an extra 64 Kbytes of read-only routines. Using this technique, with optimal library sizes, a user task can now make direct use of twice as much memory as it could have with RSX-11M. Frequently, increased task performance is gained by putting routines into an overlaid supervisor-mode library rather than overlaying the task.

RSX-11M-PLUS also supports the separate I- and D-space hardware that is available on many of Digital's processors. This means a user task can address up to 128 Kbytes of memory — 64 Kbytes of instructions and 64 Kbytes of data. Thus, when used with supervisor-mode libraries, I- and D-space tasks can address up to 196 Kbytes of memory — which can both greatly improve system performance and simplify program development.

- MEMORY MAPS
  In a mapped system, the user does not need to know where a task resides in physical memory because the map associates the task addresses with available physical memory. Memory management directives are used with the memory management hardware to perform address mapping at a level that is visible to and controlled by the user.

  The memory allocation information (map) produced by the Task Builder shows users how program sections are arranged in task memory (their starting virtual addresses and extents on mapped systems and their physical addresses and extents on unmapped systems), what attributes are in a program section, any undefined symbols, and the optional cross-reference listing of global symbols. The starting addresses, combined with the relative location counter values (provided by the assembler), can be used during debugging as offsets to access locations within the memory-resident task.

Parent–offspring Tasking

The RSX-11M-PLUS, RSX-11M, and Micro/RSX operating systems also support parent–offspring tasking. A parent task is one that starts or connects to another task, called an offspring task. This type of tasking allows simpler, more straightforward multitasking synchronization schemes and has many realtime applications.

Another major use for parent–offspring tasking is batch processing; task relationships and parameters can be set up online to control the processing of one or more batch jobs offline.

Starting (activating) offspring tasks is called spawning. Spawning also includes the ability to establish task communications. A parent task can be notified when an offspring task exits and can receive status information from the offspring task. Status returned from an offspring task to a parent task indicates successful completion of the offspring task or identifies specific error conditions.

Parent–offspring tasking also supports chaining. An offspring task can pass its parent connection to another task — thus making the new task the offspring of the original parent.

A parent task can connect to more than one offspring task. An offspring task can, in turn, be connected to more than one parent task.

Under RSX-11M-PLUS and Micro/RSX, offspring tasks can perform I/O operations with virtual terminals, just as they can perform I/O with physical terminals. A virtual terminal is not a hardware device; it is implemented in software with data structures created by the executive.

Virtual terminals are not interactive. A user does not issue requests or receive prompts; there is no immediate interaction between the user, the terminal, and the running task. Virtual terminals are used in batch processing and other offline processing environments to provide terminal I/O support for offspring tasks that normally would require user or operator intervention. For example, if a spawned task prompts for (requests) terminal input (for instance, if the spawned task is PIP), then the user needs a method of supplying the input, so it will look as if it came from a terminal. A virtual terminal can be used to supply the required input. Thus, the spawned task code is the same whether the task is spawned or initiated from a terminal.

Intertask Communication

RSX systems provide intertask communication facilities for synchronizing execution, for sending directives, and for sharing common data.

Tasks are frequently required to pass information among themselves in order to perform their functions. For instance, in a typical process control situation there may be tasks performing such functions as measuring variables (for example, temperature), controlling the process, and preparing status reports. One task may be measuring the temperature while a second task uses the temperature measurement by comparing it with a desired temperature. And a third task may be compiling a process log for inspection by the operator. One task must pass information to other tasks, in order for them to carry out their functions as required.

Intertask communications can use

- Common and group-global event flags.
- Shared data files.
- Send/receive directives (variable length on RSX-11M-PLUS and Micro/RSX).
- Shared regions.
- Virtual terminals on RSX-11M-PLUS and Micro/RSX.

- COMMON AND GROUP-GLOBAL EVENT FLAGS
  Common and group-global event flags are fast and simple. Each one conveys one bit of information suitable for communication in time-critical situations.

  Cooperating tasks can communicate using common event flags and group-global event flags. Common event flags are common to all tasks; they can be set or cleared as a result of a task's operation. Of the 32 common event flags, eight are reserved for use by the system. A task can read, set, clear, or test for common event flags to be set.

  The group-global event flags can be used in any application in which common event flags can be used; however, a group of event flags can be used only by tasks running under UICs containing the group code specified when the group-global event flags were created. A task can set, read, clear, or test for group-global event flags.

- SHARED DATA FILES
  Very large amounts of data can be shared using shared data files. Synchronization at the sending and receiving tasks may be required. The access to the data is at disk file access speeds.

- SEND–RECEIVE DIRECTIVES
  The send/receive directives can send and receive messages of moderate size at moderate speed. Synchronization is required at both the sending and receiving tasks.

  Using the send/receive directives, a task can transfer a buffer load of data to another task. The data is transferred in a 13-word block via the Dynamic Storage Region (pool) in the executive. When sent, the message is queued in the pool with other messages for the receiver task. If the receiver task is not running, its queue of messages is maintained until it runs and issues receive directives. In addition, RSX-11M-PLUS and Micro/RSX support variable-length send/receive directives that can handle up to 255 bytes.

- SHARED REGIONS
  A shared region is a block of data or code that resides in memory and can be used by any number of tasks. A shared region can contain data for use by several tasks. It can be an area in which one task writes data for use by another task. Or a shared region can contain routines for use by several tasks.

  Shared regions are a primary resource for conserving memory. In addition, they can increase productivity by allowing programmers and tasks to share data and code. There are two kinds of shared regions — resident commons, which provide a way for two or more tasks to share their data and resident libraries, which provide a way for two or more tasks to share a single copy of commonly used subroutines.

The term "resident" denotes a shared region that is built and installed into the system separately from the task that links to it. That is, programmers use the Task Builder to build a shared region much as they would build a task. Switches are used to designate the kind of shared region (a library or a common) to be built.

RSX-11M-PLUS and Micro/RSX handle shared regions dynamically. Both automatically load them into partitions and checkpoint them when they are no longer referenced. In addition, RSX-11M-PLUS has two other commonalities—shared segments of multiuser tasks and supervisor-mode libraries. Micro/RSX also supports shared segments of multiuser tasks.

- MULTIUSER TASKS
  With the Task Builder, RSX-11M-PLUS and Micro/RSX users can build multiuser tasks. In a multiuser task, one portion of its code is protected as read-only and the other portion is protected as read/write.

  Once the multiuser task is installed in memory, when any other task requests the multiuser task to run, the system duplicates only the read/write, or impure, portion of the task. The read-only, or pure, portion is shareable but hardware protected.

- SUPERVISOR-MODE LIBRARIES
  Supervisor-mode libraries, discussed above as a means of increasing a task's virtual address space, are resident libraries of routines that are used only in supervisor mode. They are available only on RSX-11M-PLUS systems running on PDP-11/44s, PDP-11/70s, MicroPDP-11/73s, and MicroPDP-11/83s.

I/O Processing

The I/O processing system consists of several highly modular interdependent components. The modular construction enables programmers to choose the programming interface and processing method appropriate for their needs without incurring runtime space or performance overhead for features not used. To maximize data throughput and minimize interrupt response time, the I/O-request-processing software also takes advantage of the hardware's ability to overlap I/O transfers with computation, to switch contexts rapidly, and to generate interrupts on multiple priority levels.

To achieve performance and space goals, the RSX I/O system centralizes common functions, eliminating the inclusion of repetitive code in every driver in the system. To do this, RSX-11 data structures are used to drive the centralized routines. This, in turn, reduces the size of the I/O drivers.

The RSX system is structured to allow entry at any level. The top level contains the File Control Services (FCS) or the optional Record Management Services (RMS). Both provide device-independent access to devices in the system.

The lowest level of task I/O is the QIO directive. Any task can issue a QIO directive. The directive allows direct control over devices that are connected to a system and that have an I/O driver.

Custom devices can be connected to the RSX-11 system through special user-written device handlers or through Connect-to-Interrupt-Vector system directives.

With the Connect-to-Interrupt-Vector system directive, a task can process hardware interrupts through a specified vector. Through a subroutine included in the task's space, the task connects to the interrupt vector and can process interrupts directly from I/O devices.

Programming Interfaces

The I/O programming tools are the File Control Services (FCS) and the Record Management Services (RMS) for general purpose file and record processing and the Queue I/O system services for direct I/O processing.

FCS and RMS (discussed in the next section) provide device-independent access to file-structured I/O devices. The most general purpose type of access enables programs to process logical records. FCS and RMS automatically provide record blocking and unblocking.

RMS users can also choose to perform their own record blocking on file-structured volumes, such as disk and magnetic tape, either to control buffer allocation or to optimize special record processing. To block their own records, users address blocks with a virtual block number (the number of the block relative to the file being processed) for volume-independent processing.

The I/O system services provide both device-independent and device-dependent programming. Users perform their own record blocking on file-structured and nonfile-structured devices. Virtual block addressing is used on Files-11 disk or ANSI magnetic-tape volumes. In addition, users with sufficient privilege can perform I/O operations using either logical or physical block addressing for defining their own file structures and accessing methods on disk and magnetic-tape volumes.

Ancillary Control Processors

FCS, RMS, and the I/O system services use the same I/O control processors. Called ancillary control processors (ACPs), they are used for processing file-structured I/O requests. An ACP provides file structuring and volume access control for a particular type of device. Typical ACP functions include creating a directory entry or file, accessing or de-accessing a file, modifying file attributes, and deleting a directory entry or file header. There are three kinds of ACPs provided in the system: Files-11 disk, ANSI magnetic tape, and network communications link.

The FCS, RMS, and I/O system services programming interfaces are the same regardless of the ACP involved but, since ACPs are specific to a device type, they do not have to be present in the system if the device is not present. There is one network ACP process for all DECnet network communications links in the system and none if the system is not in a network.

Device Drivers

Once the ACP sets up the information for file-structured I/O requests, a request can be forwarded to a device driver. All nonfile-structured I/O requests are forwarded directly to a device driver.

The functions of a driver are to

- Define the peripheral device for the rest of the RSX-11 operating systems.
- Define the driver for the operating system procedure that maps and loads the driver and its device database into system virtual memory.
- Initialize the device (and/or its controller) at system startup time and after recovery from a power failure.
- Translate software requests for I/O operations into device-specific commands.
- Activate the device.
- Respond to hardware interrupts generated by the device.
- Report device errors.
- Return data and status from the device to software.

Device drivers work in conjunction with the RSX-11 operating systems. The operating system performs all I/O processing that is unaffected by the particular specifications of the target device processing (that is, device-independent). When details of an I/O operation need to be translated into terms recognizable by a specific type of device, the operating system transfers control to a device driver that performs device-dependent processing.

The RSX operating systems contain device drivers for a number of standard Digital-supported devices. These include Q-bus and UNIBUS devices. Also, the user can write additional drivers for nonstandard devices. The *RSX-11M-PLUS* and *RSX-11M Guide to Writing an I/O Driver* manuals provide detailed information on writing, loading, and debugging drivers.

I/O Request Processing

All I/O requests are generated by a Queue I/O (QIO) Request system service. If a program requests RMS procedures, RMS issues the QIO Request system service on the program's behalf. QIO Request processing is rapid because the system can use each device as efficiently as possible by minimizing the code that must be executed, to initiate requests and post request completion anduse each disk controller as productively as possible by overlapping seeks with I/O transfers (for RSX-11M-PLUS and Micro/RSX only).

The processor's many interrupt priority levels speed interrupt response because they enable the software to have the minimum amount of code executing at high-priority levels. They do this by using low-priority levels for code-handling request verification and completion notification. In addition, device drivers take advantage of the processor's ability to overlap execution with I/O by enabling processes to execute between the initiation of a request and its completion. User tasks can queue requests to a driver at any time, and the driver immediately initiates the next request in its queue upon receiving an I/O completion interrupt.

On RSX-11M-PLUS and Micro/RSX, the driver initiates the request and returns to the executive when it is called. Disk seeks do not require the controller, once they are initiated. If a disk driver receives a seek request while the controller is busy with an I/O transfer request on another disk unit, the driver queues the request. When it has finished the current transfer, the controller will initiate the seek request before any pending I/O transfers.

RSX-11M-PLUS and Micro/RSX use optimization algorithms when choosing among I/O requests of equal priority. I/O Queue Optimization increases the throughput of the disk subsystem. For maximum flexibility, RSX-11M-PLUS and Micro/RSX provide three methods of I/O Queue Optimization. In some cases, I/O Queue Optimization results in smoother disk operation by reducing erratic head movement.

Devices

The purpose of devices of any type is to handle I/O from tasks. RSX-11M-PLUS, RSX-11M, and Micro/RSX offer users a number of ways to identify the physical devices that handle I/O:

- Mnemonics.
- Pseudodevice names.
- Logical Unit Numbers (LUNs).
- Logical device name assignments.

- MNEMONICS

  In I/O operations, the operating systems and its users deal with device drivers. Each device driver has a unique identifier consisting of a two-letter mnemonic. LP, for example, refers to the lineprinter driver, and DB is the driver for RP04, RP05, and RP06 disks.

  Individual devices are identified by the two-letter mnemonic of their driver and an octal unit number, terminated by a colon. For example, on systems with two lineprinters, one lineprinter is distinguished as LP0:, the other, as LP1:.

- PSEUDODEVICE NAMES

  A pseudodevice name is a device unit name that does not correspond to a real device until it has been redirected. These pseudodevices are forwarding addresses used by the operating system to locate real devices needed by tasks. The main function of pseudodevices is to establish relationships between devices that are tied into the system by the system generation process. Because tasks can refer to pseudodevices instead of specific physical devices, pseudodevices allow tasks a great deal of freedom in performing I/O.

  An example of how pseudodevices are useful is the commonly used pseudodevice, LB:. As part of SYSGEN, LB: is defined as the device on which the system resides. In a disk-based system, most system task-image files remain on a disk until they are needed. System task images must be installed from LB: for the system to work, but LB: need not be the same device each time. For instance, an installation with two systems, one with system-image files on DB0: (DB is the mnemonic for an RP04, RP05, or RP06 disk pack) and one with system-image files on DB1: can redirect LB: to DB0: or DB1:, depending on which system is running. Users can confidently refer in their tasks to LB: as the device from which system task images are installed, without wondering from day to day which system is running.

- LOGICAL UNIT NUMBERS

  Each task includes Logical Unit Numbers, or LUNs, that establish a relationship between the I/O done by the task and physical device units. This relationship can be different for each task.

  LUNs can be assigned by the programmer at the time the task is built, at the time the task is installed, or by the task itself at runtime. Because the system provides default LUN assignments, it is not always necessary to assign a LUN to a task. Furthermore, LUNs can be changed by a DCL or MCR command for any installed, inactive, nonfixed task.

- LOGICAL DEVICE NAMES

A LUN is simply a name used to represent the relationship between a physical device and a logical device name. Logical device names provide a means by which tasks can maintain device independence. Logical device names have the same syntax as other device names. The logical device name can be the same as a standard RSX-11 device or pseudodevice, such as LP0: or LB1:, or it can use two letters with a user-assigned meaning, such as AZ:.

There are three types of logical device assignments.

- *Local assignments*, which can be made by any user, apply to tasks initiated from the terminal used to make the assignment. Local assignments override both other categories of assignments. Different users can assign the same local logical name to different devices.
- *Login assignments* are controlled by privileged users through ACNT, the Account File Maintenance Program, or through ASSIGN/LOGIN, a privileged command. Login assignments are reestablished each time the user logs in. They override the next category global assignments and are used only in systems with multiuser protection. Login assignments remain in effect until the user logs off or until a privileged user deassigns them.
- *Global assignments*, which can be made only by privileged users, apply to all tasks running in the system.

File Structures and Access Methods

The RSX-11M-PLUS, RSX-11M, and Micro/RSX operating systems provide integrated support for file management and access through the following mechanisms:

- The Files-11 file system.
- File and volume management utilities.
- File Control Services.
- Record Management Services.

The file system provides volume structuring and directory access to disk and magnetic tape files. Programmers can use the file system as a base for building their own record processing system, or they can use the FCS File Control Services or RMS Record Management Services.

Both FCS and RMS provide device-independent access to all types of I/O peripherals. The FCS and RMS procedures enable a program to access records within files and provide the same programming interface, regardless of device characteristics. The system includes utilities for RMS file creation and maintenance.

FCS enables users to perform record-oriented and block-oriented I/O operations and to perform additional functions for file control. It supports both sequential and random access to data in files on sequential-access devices (such as magnetic tapes) and random-access devices (such as disks).

RMS, developed after FCS, allows more complex file organizations than are possible with FCS. With it, programmers have a wider choice of file organizations and record access modes and retrieval capabilities. They can use sequential, relative, or multikey indexed-sequential file organizations, and sequential, random, or record's file address access modes.

File Management

RSX systems include Files-11 for overseeing the storage and handling of files on volumes. Volumes contain both user files and system files. Files-11 accepts both DCL and MCR commands for initializing (or formatting) the three types of volumes—disks, DECtapes, and magnetic tapes.

Volumes that are not in Files-11 format are described as foreign. Although Files-11 cannot access foreign volumes directly, FLX, the File Exchange program, translates files from other Digital formats to Files-11 formats.

Volume and file protection are based on User Identification Codes (UICs) assigned to accessors and to the file or volume. The UICs establish the accessor's relationship to the data structure as either owner, owner's group, system, or world (all others). Depending on the relationship, the accessor may or may not have read, write, extend, or delete access to any given file.

The file structure on disk volumes appears to a program to be a virtually contiguous set of blocks. The blocks of the file, however, can be physically located anywhere on a volume. Mapping information is maintained to identify all blocks that constitute a file.

Magnetic tape and DECtape contain files that consist of physically contiguous blocks. These files have ANSI-format labels.

▪ FILE DIRECTORIES AND DIRECTORY STRUCTURES

A directory is a file containing a list of files on a given volume. A directory entry contains the name, type, version, and file identifier for a particular file. A directory can list files having the same owner UIC or files having different owner UICs.

A disk volume contains at least one directory, called the Master File Directory (MFD), that contains a list of user directory files. The utility that initializes volumes creates a volume's MFD. When a user creates a file, the system places the filename in a User File Directory (UFD) and stores the user's current UIC in the file header to indicate the owner of the file. All UFDs are listed in each volume's MFD. UFD and MFD directories list the names of files and contain pointers to each file's header. The file header contains information about the file's owner and the physical location of the file segments.

A task or user must satisfy the protection mask of both the file to be accessed and the UFD in which the file is located, in order to gain access to a file.

Because directories of files are files themselves, they are assigned owner UICs and can be protected from certain kinds of access, depending on the relationship established by a user's UIC. In the special case of directory files, the file protection fields control an accessor's ability to

- Look up files.
- Enter new files in the directory, including new versions of existing files.
- Remove files from the directory.

- FILE SPECIFICATIONS

A file specification identifies the file to be used in a file-processing operation. Programs use file specifications to identify the file they want to create, access, delete, or extend. Users supply the command interpreter with a file specification to identify the file they want to edit, compile, build a task, copy, or delete, for example. A file specification can be composed of a

- *Device name*—the physical or logical device unit on which the volume containing the file is mounted. The device name is followed by a single colon to delimit it from the remainder of the file specification.
- *User Identification Code (UIC)*—a specification for the User File Directory (UFD) in which the file is listed. The UIC represents the owner's group and the member number and serves as a protection for files and directories. The UIC is always enclosed in brackets. Micro/RSX and RSX-11M-PLUS also support the use of named directories similar to those used by VAX/VMS, except that hierarchical directories are not supported.
- *Filename*—an alphanumeric string from one to nine characters long that specifies the name of the file. A period always separates the filename from the filetype.

- *Filetype* — a three-letter alphanumeric string that often is a mnemonic that identifies the nature of the file contents. A semicolon always separates the filetype from the version number.
- *Version number* — an octal number that indicates the version or generation of the file.

An example of a complete file specification is

DK0:[200,200]TECSUM.TXT;2

In this case, DK0 is the name of the device; [200,200] is the UIC; TECSUM is the filename; TXT is the filetype; and 2 is the version number.

Normally, users don't have to provide a complete file specification to identify files. The system sets defaults for most fields in a command-line file specification, depending on the program. When users omit one or more fields in a file specification, the system assumes the default. For example, if the device name is not present, the device is assumed to be the user's current system device. If the version number is not present, it is always assumed to be the latest.

Sometimes users can specify more than one file in a single specification by using a wildcard in one or more fields of the specification. The wildcard causes the system to select all the files that satisfy the explicitly described fields. This saves both keystrokes and time.

- RSX-11M-PLUS AND MICRO/RSX ALSO SUPPORT THE USE OF LOGICAL NAMES

A logical name is a user- or system-defined name for an equivalence name that can be the following:

- All or part of a file specification — This keeps your programs and common procedures independent of physical file specifications.
- A physical device — You can assign logical names to devices such as magnetic tape drives, terminals, and lineprinters. The system manager can assign logical names to public disk volumes, so that users do not have to be concerned with the physical location of these volumes.
- Anything created by the DEFINE command.

In addition, to reduce typing, you can use logical names as a shorthand way of specifying files or directories that you refer to frequently. For example, you might assign the logical name HOME to your default disk and directory, or the logical name DIARY to a file in which you keep a log of your daily activities. You can also use logical names in file specifications to keep your programs and command procedures independent of physical file specifications.

Another common use for logical names is to refer to physical devices. For example, you can assign logical names to devices such as tape drives, terminals, and lineprinters. Also, the system manager or a privileged user can assign logical names to public devices, so that the users do not have to be concerned with the physical location of those volumes.

File and Volume Management Utilities
RSX operating systems provide many services that aid in file and volume management and maintenance. Programs to locate bad blocks, to verify file structures, and to backup and restore files and volumes are among these services. Refer to other sections in this chapter that discuss System Management and Maintenance and System Utilities.

File Control Services
RSX File Control Services (FCS) is a set of routines that a task can use to access the file system. It enables users to conduct record-oriented and block-oriented I/O operations and to perform other functions required for file control, such as creating, deleting, opening, closing, reading, and writing. To use FCS, a task invokes the FCS macros. The macros call FCS routines, which issue the actual I/O directives (QIOs). Most of the RSX-11M-PLUS, RSX-11M, and Micro/RSX tasks and utilities use FCS.

The FCS routines, which can reside in the task's image, in a resident system library, or in a system object-module library, are linked with a task when the task is built. These routines, consisting of pure position-independent code, provide a user interface to the file system, enabling the user to read and write files on file-structured devices and to process files in terms of logical records.

FCSRES is a resident library of commonly used FCS routines. Users can build tasks to link to this single copy of the FCS routines instead of including the routines in each task image. Having only one copy of the FCS routines reduces memory usage. A task accesses a resident library by using the user-mode mapping registers. These registers are also used to map the task code and data, so each task that uses the FCSRES library must reserve some of its logical address space to map the library.

The FCSRES library uses no special hardware and can thus be used on any of the processors that RSX-11M-PLUS, RSX-11M, and Micro/RSX support.

RSX-11M-PLUS and Micro/RSX systems support FCSFSL, a supervisor-mode library of commonly used FCS routines. Tasks built with FCSFSL have exactly the same capabilities as tasks built with FCSRES or tasks that include the FCS routines in their task images.

A task accesses a supervisor-mode library using the supervisor-mode mapping registers, a hardware feature available on certain newer PDP-11s and MicroPDP-11s. Mapping the library in supervisor mode allows the library to reside outside the task's logical address space. Tasks built with FCSFSL can therefore be larger than the same tasks built with FCSRES.

With FCS, users can write a collection of data (consisting of distinct logical records) to a file in a way that enables them to retrieve the data at will. Data can be retrieved from the file even when the user doesn't know the exact form in which the data was written to the file. FCS thus provides a sense of transparency to the user so that records can be read or written in logical units that are consistent with an application's requirements.

- FILE ACCESS METHODS

  Under FCS, RSX systems support both sequential and random access to files. The sequential access method is device-independent; that is, it can be used for both record-oriented and random-access devices (magnetic tapes and disks, for example). The direct access method can be used only for file-structured, random-access devices.

- DATA FORMATS FOR FILE-STRUCTURED DEVICES

  Data is transferred between peripheral devices and memory in blocks. A data file consists of virtual blocks, each of which can contain one or more logical records. Records in a virtual block can be either fixed or variable in length.

  Virtual blocks and logical records within a file are numbered sequentially, starting with one. A virtual block number is a file-relative value, while a physical block number is a volume-relative value. For example, the first virtual block in a file is always virtual block number 1, but at the same time it could also be physical block number 156.

- BLOCK I/O OPERATIONS

  The READ and WRITE macro calls allow the user to read and write virtual blocks of data from and to a file without regard to logical records in a file. Block I/O operations provide a very efficient means of processing file data because such operations do not involve the blocking and deblocking of records within the file. Also, in block I/O operations, the user can read or write files in an asynchronous manner; control can be returned to the user program before the request I/O operation is completed.

  When block I/O is used, the number of the virtual block to be processed is specified as a parameter in the appropriate READ and WRITE macro call. The virtual block so specified is processed directly in a buffer reserved by the program in its own memory space.

As implied above, the user is responsible for synchronizing all block I/O operations. For this purpose, the user invokes a WAIT$ macro when it is necessary to wait for the I/O to complete. A user-selectable event flag can be used to coordinate block I/O transfers, allowing simultaneous asynchronous I/O to a number of files.

- RECORD I/O OPERATIONS

The GET$ and PUT$ macro calls are provided for processing record-oriented files. GET$ and PUT$ operations perform the necessary blocking and deblocking of the records within the virtual blocks of the file, allowing the user to read or write individual records.

When writing a new file with PUT$ operations, the user program must specify the format of the records. For example, it must specify whether the records are fixed or variable in length, or whether records that are to be output to a carriage-control device are to contain carriage-control information, which can be either at the beginning of the records or embedded within the records.

For sequential access files, I/O operations can be performed for both fixed- and variable-length records. For direct access files, I/O operations can be performed only for fixed-length records.

In contrast to block I/O operations, all record I/O operations are synchronous; control is returned to the user program only after the requested I/O operation is performed.

However, FCS has facilities for simultaneous I/O to a ring of buffers, called multiple buffering, and I/O to a buffer of several disk blocks, called big buffering, to increase program efficiency at the cost of using additional virtual memory.

Because GET$ and PUT$ operations process logical records within a virtual block, only a limited number of GET$ or PUT$ operations result in an actual I/O transfer, that is, when the end of a data block is encountered. Therefore, not all GET$ and PUT$ I/O requests necessarily involve a physical transfer of data.

- THE FILE STORAGE REGION

The file storage region (FSR) is an area allocated in the user program as the working storage area for record I/O operations. The FSR consists of two program sections that are always contiguous. The first program section of the FSR contains the block buffers and the block buffer headers for record I/O processing. The user determines the size of the area at assembly time. The number of block buffers and associated headers is based on the number of files that the user intends to open simultaneously for record I/O operations.

The second program section of the FSR contains impure data that is used and maintained by FCS in performing record I/O operations. Portions of this area are initialized at the time the task is built, and other portions are maintained by FCS. This program section is intentionally isolated from the user to preserve its integrity.

Blocking and deblocking of records during input is accomplished in the FSR block buffer during output. Note also that FCS serves as the user interface to the FSR block buffer pool. All record I/O operations initiated through GET$ and PUT$ calls are totally synchronized by FCS.

- DATA TRANSFER MODES

  When record I/O is used, a program can gain access to a record in either of two ways after the virtual block has been transferred into the FSR from a file.

  - *Move mode.* Individual records are moved from the FSR buffer. Move mode simulates the reading of a record directly into a user record buffer, thereby making the blocking and deblocking of records transparent to the user.
  - *Locate mode.* The user program accesses records directly in the FSR block buffer. Program overhead is reduced in locate mode because records can be processed directly within the FSR block buffer.

- SHARED ACCESS TO FILES

  FCS permits shared access to files according to established conventions. Two macro calls, among several available in FCS for opening files, invoke these functions. The OPNS$ macro call is used specifically to open a file for shared access. On the other hand, the OPEN$ call invokes generalized open functions that have shared access implications only in relation to other I/O requests then issued.

  By default, several tasks can read the same task simultaneously. Restricted or shared access to files being read or written is possible. Shared access during reading does not necessarily imply the presence of read requests from several separate tasks. The same task can open the same file using different logical unit numbers.

- SPOOLING OPERATIONS

  FCS provides facilities at both the macro and subroutine level to queue files for subsequent printing. A task issues the PRINT$ macro call to queue a file for printing on the system lineprinter.

- FCS MACROS AND MACRO USE

  FCS includes four basic kinds of macros that simplify the user's interface to the system's file control primitives. The four kinds are initialization macros, file-processing macros, command-line-processing macros, and the CALL macro.

The initialization and file-processing macros are used to establish the database description and the necessary temporary storage areas needed to perform I/O operations. The command-line-processing macros are used to dynamically process I/O commands entered from a terminal. The CALL macro is used to invoke file-control routines.

The initialization and file-processing macros set up the following structures to define the database:

- A file data block (FDB) that contains execution-time information necessary for file processing. It defines the basic characteristics of a file — for example, record type, record size, and access privileges.
- A data set descriptor that is accessed by FCS to obtain the file name, type, version number, and location necessary to open a specified file. The data set descriptor is used when a program accesses a given set of known or predefined files.
- A default file name block that is accessed by FCS to obtain default file information required to open a file. This is accessed when complete file information is not specified in the data set descriptor. It is used by programs written to access a general set of files.

There are two types of initialization macros — assembly time macros and runtime macros. Data supplied during assembly of the source program establish the initial values in the FDB. Data supplied at runtime can either initialize additional portions of the FDB or change values established at assembly time. Furthermore, the data supplied through the file-processing macros can either initialize portions of the FDB or change previously initialized values. The user not only has a broad range of control over defining the database characteristics but also has control over when the definitions are made.

File-processing macros also determine the way in which files are processed. These macro calls are invoked and expanded at assembly time. The resulting code is then executed at runtime to perform various file-processing operations.

The file-processing macros allow the user to specify random access or sequential access to files and to perform block-oriented or record-oriented file processing. In addition, the PRINT$ macro allows the user to spool files to a lineprinter or terminal device.

The command-line-processing macros allow the user to access special routines available in the system object library. The Get Command Line (GCML) routine accomplishes all the logical functions associated with the entry of a command line from a terminal, an indirect command file, or an online storage medium. The Command String Interpreter (CSI) routine takes command lines from the GCML input buffer and parses them into appropriate data set descriptors required by FCS for opening files.

The CALL macro allows the user to access a special set of file control routines. Among other operations, these routines allow a MACRO program to perform the following: find, insert, or delete a directory entry; rename a file; extend a file; mark a temporary file for deletion; and delete a file.

Record Management Services

Record Management Services (RMS), a set of general purpose file-handling capabilities, combines with the host RSX-11M-PLUS, RSX-11M, or Micro/RSX operating system to provide efficient and flexible data storage and modification. When writing programs, users can select processing methods suitable to their application from among several RMS file structuring and accessing techniques.

Not only does RMS handle such functions as file organization and access methods but it also manages the other file attributes (for example, storage medium and record format) and the runtime environment. By accomplishing most of its work transparently, RMS relieves programmers of many of the complexities associated with file and record manipulation.

Included in RSX-11M-PLUS, RSX-11M, and Micro/RSX, RMS is also part of all operating systems available from Digital for the PDP-11 family. For more details, refer to Chapter 18, "Record Management Services."

▪ Security Features

A number of terminals can operate concurrently when using an RSX operating system. Each user terminal operates independently of others in the system so that each can run a different task and each can run more than one task. In a system that supports multiuser protection, a user must log onto the terminal before issuing commands.

To provide system security and authorize privileges in a multiuser environment, RSX systems include a protection scheme based on User Identification Codes (UICs). System management assigns each user a UIC that consists of two numbers: a group number and a member number. The UIC also determines whether the user is privileged or nonprivileged. Privileged users have access to every part of the operating system. Nonprivileged users can use most of the operating system, but they cannot change it.

When a file, program, or intertask communication facility is created, the system places the filename in a User File Directory (UFD) and stores the creator's UIC in the file header to indicate the owner.

The file's UIC determines which groups of users or programs have controlled access to the file. Every file also has a protection code that specifies what the users may do to the file when they access it. Either the file's owner or a privileged user sets and can change the file's protection code.

To access a file, a user must know the UFD in which it is listed. This knowledge, however, is not sufficient to guarantee access. The user must satisfy conditions specified in a protection code associated with the file to be accessed. In addition, the volume that contains the file must be mounted before the file is accessed.

RSX-11M-PLUS, RSX-11M, and Micro/RSX allow four types of access:

- *Read.* The user or the user's task may read, copy, print, or type the file and, if it is a task, run it.
- *Write.* The user or the user's task may add new data to the file by writing to space already allocated to the file.
- *Extend.* The user or the user's task may change the amount of disk space allocated to the file.
- *Delete.* The user or the user's task may delete the file.

The four user groups are defined by their UICs:

- *System.* Every user or program whose UIC group number is a system-privileged group number (group number less than 10, octal).
- *Owner.* The user whose UIC is the same as the UIC assigned to the file.
- *Group.* Every user whose UIC group number is the same as that assigned the file.
- *World.* All other users.

RSX-11M-PLUS and Micro/RSX also support user password encryption. A one-way encryption algorithm similar to that used by VAX/VMS encrypts a user's password after it is entered. Passwords stored by the system are maintained in encrypted form. Anyone gaining access to the encrypted password database and knowing the encryption algorithm still will not be able to decrypt the password.

Integrity/Reliability Features

The RSX operating systems include a variety of aids that help operators ensure system integrity. These are both comprehensive and easy to use.

I/O Exerciser

The I/O Exerciser (IOX) detects and diagnoses I/O problems on the disks and tapes in a system's hardware configuration. IOX exercises Files-11 disks, nonfile-structured disks, magnetic tapes, DECtapes, and cassettes. Used by system management and maintenance personnel, it determines whether these units are correctly executing I/O operations. In addition, IOX measures system activity and provides a command language for executing test functions. IOX output consists of detailed error reporting and general information that describes system activity.

IOX performs three kinds of exercises. Operators use the IOX Command Language to specify and control the exercise for the units in a system. They choose an exercise appropriate to a unit and they set exercise parameters that determine how the unit is exercised.

Error Logging

The RSX11M-PLUS, RSX-11M, and Micro/RSX Error Logging Systems record information about errors and events that occur on system hardware, either for immediate action or for eventual analysis and reporting. Error Logging handles mass-storage device (disk and tape) errors, as well as memory errors. Because Error Logging is a part of all three systems, it is most effective for detecting hardware errors that allow the system to continue functioning.

Error Logging is not used to detect information about operating-system failures or about device problems that cause the system to fail. However, it does provide information about what I/O activities occurred on a device at the time of an I/O failure. If a system includes the Crash Dump Analyzer (CDA), CDA can provide reports on operating-system failures.

Error Logging Reports can be used to determine that a device is having problems before it actually fails and causes lost data. For example, a report showing a pattern of recurring errors from different blocks on a single disk head may indicate that the head needs to be replaced.

Crash Dump Analyzer

The Crash Dump Analyzer (CDA) is a specialized utility that helps establish the cause of system crashes. It reads the contents of a system memory dump, formats the information, and outputs the following type of information to a lineprinter for evaluation:

- All memory management and hardware register contents.
- The system stack.
- Task control blocks for each active task.
- Contents of the clock queue.
- Information on all devices in the system.
- Contents of physical memory.
- Task headers for each task in memory.
- Contents of each Partition Control Block.
- Contents of the system Dynamic Storage Region (pool).
- Contents of the System Task Directory for all tasks.

The CDA is a valuable tool for helping to eliminate a variety of causes of system crashes, because system crashes can occur when users make modifications to the executive or when privileged tasks are mapped into the system data structures, but are not yet debugged.

Online Software Maintenance

Another way the system manager or operator maintains system integrity is by using the software maintenance tools that make patches and install updates to the RSX binary and source code. *UPDATE* uses a machine-readable format; the SYSGEN process updates the system automatically with the updates supplied.

Remote Diagnostics

On PDP-11/44 systems equipped with the remote diagnosis option, the system manager can set up the system for remote preventive maintenance or troubleshooting. When a hardware problem is detected or suspected, the system manager mounts a diagnostic disk pack, sets a switch on the processor console, and calls Digital's local service office. A system manager need not be present at the installation, once the call is made. A technician at Digital's diagnostic center can then connect to the installation, run automated diagnostics, operate the diagnostic console manually, and check the error-log file. If a problem is found, one of Digital's Field Service engineers can bring the proper equipment and replacement modules to make repairs.

RSX-11M-PLUS Reconfiguration Services

In a reconfigurable system, resources such as memory and devices can be added or removed. Reconfiguration is a means of physically and logically connecting and disconnecting various resources. Reconfiguration services are available only on RSX-11M-PLUS systems.

With the reconfiguration services, an RSX-11M-PLUS system can be reconfigured to bypass faulty hardware elements and to isolate the system from the effects of these elements. An operator can reconfigure a system interactively from a terminal or by means of indirect command files. The reconfiguration services act as an interface between the operator's terminal and the RSX-11M-PLUS system.

An operator can define a set of hardware resources that are accessible from the online system and also remove devices from the pool of resources allocated to the online system. For example, after booting the system, the operator can place a failed disk drive offline and then use another drive — either one already online or one placed online specifically for this purpose — to take over the disabled unit.

Reconfiguration services allow faulty hardware to be isolated so that it does not affect the system — and system users — adversely.

Shadow Recording

With Shadow Recording, an RSX-11M-PLUS system backs up all new data as it is written to a Files-11 disk. A SYSGEN option, Shadow Recording, creates two identical sets of disks that are called a "shadowed" pair. More than one pair of disks can be shadowed, but shadowed pairs cannot overlap. The two disks must be of the same type — both RK02s or both RA80s, for example. The first disk of the pair, the primary disk, is the original disk that exists whether or not Shadow Recording is active. Any disk on an RSX-11M-PLUS system, including the system disk, can be the primary disk of a shadowed pair. The second disk of the pair, the secondary disk, becomes an exact copy of the primary disk.

Shadow Recording operates transparently — writing to and reading from the secondary disk as it writes to the primary disk. When a disk read occurs, the executive reads the secondary disk. The executive displays all I/O errors occurring on a Shadow Recording disk pair on the operator's console.

Shadow Recording provides a dynamic backup of all blocks as they are written to the primary disk. That is an important feature for many processing environments, particularly environments in which critical information must be duplicated to safeguard against inadvertent damage or loss in the event a disk error occurs. Recovery may be quicker too, and downtime may be reduced, because disk errors do not necessarily mean an application must be halted.

Because it does backup online and provides an "instant" duplicate disk, Shadow Recording can be used with systems where later backup time or resources are unavailable.

Power Failure Automatic Restart

Power failure automatic restart is the ability of a system to smooth out intermittent short-term power fluctuations with no apparent loss of service and without losing data, while maintaining logical consistency within the system itself and the application tasks.

When the power begins to fail, the processor traps to the executive, which saves all register contents. When power is restored, the executive again receives control and restores the previously preserved state of the system.

The executive then informs any tasks that have requested power failure restart notifications through the Asynchronous System Trap mechanism that a power failure has occurred. These tasks can then, if required, make the restorations of state deemed necessary. The executive calls all device drivers that were active at the time the power failure occurred at their powerfail entry point. Drivers have the option of always being called on power recovery or of being called only when the driver has outstanding I/O. These drivers can then, if required, make those restorations of state (for example, repeating I/O requests) that they deem necessary. This approach is quite efficient because the repeating of I/O is placed nearest the source most likely to contain instructions on how to make the restoration.

▪ Performance Features

On RSX systems, users have the tools they need to fine-tune their systems for maximum performance and flexibility.

Pool Monitoring

Pool monitoring support controls the use of the system's Dynamic Storage Region (DSR or pool). Pool is a contiguous area in memory allocated by the executive and used as a workspace for storing such system data structures as system lists and control blocks. Pool is included at the top of the executive's permanently mapped address space in memory. The size of pool must be sufficient to handle all the dynamic storage requests of the system. By default, SYSGEN extends the mapped system executive to its maximum size to create the largest possible pool space.

Pool requirements for a system depend on the configuration, application, and degree of system loading. Enough pool must be available to satisfy peak demands; otherwise, system performance will be degraded.

Because nearly all executive functions require pool, a system can exhaust pool when system activity is very heavy. This can happen if too many tasks are installed, if too many volumes are mounted, or certain other conditions are present. When this happens, the system does not appear to have crashed, but it is not functioning normally.

To avoid this condition, pool monitoring can be used. Pool monitoring oversees pool levels, restricts use, and notifies the operator if pool is near depletion. Pool monitoring has two components — the RSX-11M-PLUS executive pool monitor code and the privileged Pool Monitor Task (PMT).

The pool monitor code within the executive monitors the amount of free pool and detects major pool events. When a major pool event occurs, the executive notifies PMT.

PMT's response to the information provided by the executive depends on specific pool events and conditions. It executes actions appropriate to the condition detected. For example, for a low-pool state, PMT establishes pool-use controls, including preventing nonprivileged users from logging on and suppressing INSTALL/RUN/REMOVE sequences on nonprivileged terminals. It also sends warning messages to all logged-on terminals and displays pool statistics at the console terminal.

Pool-monitoring support is available on mapped systems only. It is a SYSGEN option on RSX-11M systems and it is included by default on RSX-11M-PLUS systems.

I/O Queue Optimization

To increase the throughput of disk subsystems, RSX-11M-PLUS and Micro/RSX support I/O Queue Optimization, a set of optimization algorithms that both systems use when choosing among I/O requests of equal priority. For maximum flexibility, system management can choose from three methods of I/O Queue Optimization.

- The *nearest cylinder* method processes the I/O request closest to the current request, regardless of whether it comes before or after the current request.
- The *elevator* method processes I/O requests as it moves in one direction along the disk, then changes direction to process requests in the opposite direction.
- The *cylinder scan* method processes requests in only one direction along the disk, the direction being from the lowest cylinder number to the highest.

System management can choose the method that best suits the processing environment, the physical location of data on a disk drive, and how often tasks access data areas. All three methods attempt to minimize head-seek time, which decreases the seek-time component of the total file service time.

Without I/O Queue Optimization, RSX-11M-PLUS and Micro/RSX group the I/O requests in the queue, by priority, and on a first-in/first-out basis. The highest priority requests appear first in the queue and are processed in sequence. With I/O Queue Optimization, the I/O requests within priority groups are examined, and the request having the "best" disk address is processed first. Highest-priority requests are still serviced before lower-priority requests. However, throughput is enhanced by advantageous reordering of requests within priority levels. A "fairness count" is also set that limits the number of times an I/O request can be passed over.

By reducing erratic head movement, I/O Queue Optimization has the potential to speed disk operation.

Cluster Library Support

With cluster libraries, only the library in use at any one time is actually mapped to the task address space rather than the simultaneous mapping of all libraries. This leaves more task address space for application code.

The term "cluster libraries" refers to both a function and a structure created by the Task Builder that allow a task to dynamically map memory-resident shared regions at runtime. Cluster libraries permit a task to use, for example, a F77CLS library, an FMS-11 library, and an FCS-11 library, all mapped through the same task address window. The runtime routines put into the task by the Task Builder remap the library regions so that, instead of occupying 48 Kbytes of virtual address space, they share 16 Kbytes of virtual address space.

RSX-11M-PLUS and Micro/RSX Performance Enhancements

RSX-11M-PLUS and Micro/RSX contain many performance and throughput enhancements. Memory is most efficiently used by separating pure (read-only) and impure (read/write) sections of tasks. When the first copy of a multiuser task is executed, one copy of pure and impure sections is loaded. Successive users receive only a copy of the impure code, and all users share the pure section, thereby using memory more efficiently.

On RSX-11M-PLUS and Micro/RSX, shared libraries, pure sections of multiuser tasks, and common data areas can be checkpointed to disk when not being used. These tasks are automatically brought back into memory when an executing task references them. Shared areas can also be shuffled to compact memory and to remove space between tasks.

RSX-11M-PLUS and Micro/RSX users can effectively triple the memory directly addressable by tasks on certain PDP-11 and MicroPDP-11 systems through the use of supervisor-mode library routines and separate user-mode Instructions and Data (I- and D-) space. Because this increased address space means programs can be written that require fewer (if any) overlays, faster program execution and decreased program development and task-build time result.

- USER-MODE I- AND D-SPACE

  A conventional RSX-11M-PLUS or Micro/RSX task operating in user mode (as most user tasks do) can contain 64 Kbytes of virtual address space and access approximately 64 Kbytes of physical memory. However, a task using both user-mode I- and D-space Active Page Registers (APRs) can contain 128 Kbytes of virtual address space and access approximately 128 Kbytes of physical memory. RSX-11M-PLUS and Micro/RSX support the separate I- and D-space hardware available on PDP-11s and MicroPDP-11s with the appropriate hardware support.

  A conventional task running in an I- and D-space system uses both sets of APRs. However, the relocation addresses in both I-space and D-space APRs are identical. Also, the task windows refer to I-space and refer to I-space APRs in a task that does not use D-space.

  An I- and D-space task can use separately both I- and D-space APRs; that is, APRs used in this way are not overmapped. Because of this, the task can use eight D-space APRs to access and use data and eight I-space APRs to access a total of 128 Kbytes of physical memory at one time.

- SUPERVISOR-MODE LIBRARIES

  Supervisor-mode libraries are shareable resident routines that are used only in supervisor mode. A task switches to supervisor mode when it calls a routine within the supervisor-mode library. By using this technique alone, a task's virtual address space can be doubled to 128 Kbytes.

  Supervisor mode is one of three possible modes of operation (user and kernel are the other two) in which certain PDP-11 and MicroPDP-11 processors can operate. Each mode has associated with it 16 Active Page Registers (APRs) — eight I-space and eight D-space APRs.

  Normally, a task has an address space of 64 Kbytes by using the eight user-mode APRs. When a conventional RSX-11M-PLUS or Micro/RSX task links to a supervisor-mode library and calls a routine in the library, the executive copies the user-mode I-space APRs into the supervisor-mode D-space APRs and maps the supervisor-mode library with I-space APRs. Therefore, while in supervisor mode and within the library, the task can access 64 Kbytes of its own space with the D-space APRs and 64 Kbytes of library routines with I-space APRs.

  In an I-and D-space task, mapping is done differently. When an I-and D-space task links to a supervisor-mode library, the executive copies the user-mode D-space APRs into the supervisor D-space APRs. Therefore, the supervisor-mode routines can access user-mode data space and access supervisor-mode instruction space with supervisor-mode I-space APRs.

On systems with I- and D-space hardware, RSX-11M-PLUS and Micro/RSX also use separate hardware memory mapping for the executive code and executive pool. Pool is the memory available to applications for dynamic data structures. The separation of data and the use of extra memory management registers allow RSX-11M-PLUS and Micro/RSX to devote a full 40 Kbytes to system pool (minus the size of the system I/O and Micro/RSX data structures). This, in addition to the amount of system pool that has been freed by the creation of a secondary pool area, means that RSX-11M-PLUS and Micro/RSX often double the amount of pool in contrast to comparable RSX-11M systems. So RSX-11M-PLUS and Micro/RSX can support about twice as many terminals and active tasks as RSX-11M systems.

- OTHER RSX-11M-PLUS AND MICRO/RSX PERFORMANCE ENHANCEMENTS
RSX-11M-PLUS and Micro/RSX also support disk data caching and overlapped disk seeks to improve performance of typically I/O-bound applications. Disk data caching reduces the number of disk accesses required in a given application. Overlapped disk seeks allow more disk accesses per unit of time.

▪ System Utilities

System utilities help various levels of system users to interact with the system, develop applications interactively, and manage and control system resources. Through these utilities, privileged users can monitor and control system use, tune system performance, and supervise day-to-day operations. Nonprivileged users can interact easily with the system to maintain files and enter, validate, and extract data. Fast, compact, interactive editors, online debuggers, and dump analyzers make program development fast and easy.

Text Editors
For fast and easy program data entry and modification, RSX-11M-PLUS, RSX-11M, and Micro/RSX provide the EDT text editor, and on some older systems, EDI. Both feature automatic backup of input files so that if a user accidentally deletes a large amount of text, or if the equipment fails, the latest backup file is available for quick recovery.

The user invokes the EDT and EDI text editors interactively; that is, the user creates and processes files online.

- EDT EDITOR

EDT, the standard Digital editor, is an interactive text editor that is particularly useful for entering and maintaining text files. EDT is available on many Digital operating systems and is especially powerful when used on Digital's VT200 and VT100 videoterminals because it can take advantage of the editing keypads on these terminals. With keypad mode, a single keystroke performs an entire editing function—for example, deleting, reinserting, or replacing a word. Users can even redefine the functions of keypad keys (through key macros) to produce commonly used operations.

In change mode on a VT200 or VT100, users can edit one 22-line window (screenful) at a time so that they can observe immediately the effects of any editing operations performed. Instead of being restricted to the most recently altered line, users can see a whole screenful of text and can see the relationship of new and old lines. If the text is longer than 22 lines, it can easily be scrolled through to get to any other point in the file.

EDT has two features that distinguish it from EDI:

- It provides unlimited access to an entire file at one time, making it unnecessary to work with smaller sections of a file, as is usually necessary with EDI.
- It provides character-mode editing for users with videoterminals. Character mode allows editing at the character and word level, as well as at the line level.

- EDI EDITOR

EDI is a single-pass, line-oriented, interactive editor that is used to create and maintain text and source files. EDI accepts over 50 commands that determine its mode of operation and control its actions on input files, output files, and working text buffers.

EDI functions on a line-by-line basis. In block-mode operation, EDI reads a block of text from the disk file. A block contains only that amount of text that will fit into the EDI buffer; editing operations are performed on the text that is in the buffer. When work on that block is completed, a user requests the editor to renew the buffer with the next block of text.

Program Development and Maintenance Utilities

Several kinds of utilities are helpful in program development. These utilities are programs that allow programmers to work with different kinds of files and the contents of those files. Utilities can be invoked from either the Digital Command Language (DCL) or the Monitor Console Routine (MCR) environment.

- FILE MANIPULATION UTILITIES

  Digital provides many file manipulation utilities, two of which are the Peripheral Interchange Program (PIP) and the File Transfer Program (FLX). With these utilities, users can, among other jobs, copy and spool files and transfer files between volumes.

  PERIPHERAL INTERCHANGE PROGRAM - PIP transfers data files from one standard Files-11 device to another. PIP also performs file control functions. Some of the functions PIP performs are

  - Copying files from one device to another.
  - Deleting files.
  - Renaming files.
  - Listing file directories.
  - Setting the default device and UIC for PIP operations.
  - Unlocking files.
  - Spooling files.
  - Specifying file protection values.

  FILE TRANSFER PROGRAM - FLX transfers files with different formats from one volume to another. In addition, FLX converts the format of the transferred file to conform to the format of the target volume.

  FLX allows users to

  - List directories of cassettes, Digital's RT-11, or DOS-11 volumes.
  - Delete files from DOS-11 and RT-11 file-structured volumes.
  - Initialize cassettes, RT-11, or DOS-11 volumes.

  FLX performs file transfers and format conversions as appropriate from

  - DOS-11 to Files-11 volumes.
  - Files-11 to DOS-11 volumes.
  - DOS-11 to DOS-11 volumes.
  - Files-11 to Files-11 volumes.
  - Files-11 to RT-11 volumes.
  - RT-11 to RT-11 volumes.
  - RT-11 to Files-11 volumes.

File Spooling Utilities

File spooling (Shared Peripheral Operations Online) support for RSX-11M-PLUS and Micro/RSX includes the Queue Manager and batch processing. RSX-11M can include either the Queue Manager or the more primitive Serial Despooler Task; the choice of either tool is made at system generation.

Spooling on RSX-11M-PLUS, RSX-11M, and Micro/RSX is gathering output on a mass-storage device—usually a disk—to be sent to an output device—particularly a lineprinter—in an orderly fashion. Despooling is the orderly transfer of this output from the mass-storage device to the output device.

Users can spool files by using the PRINT command. Files spooled by tasks will also be queued automatically. With command qualifiers, attributes can be set for the job, and the queues can be displayed. Users can alter, hold, or release a job after it has been placed in a queue.

The Queue Manager is a collection of programs that provides for the orderly processing of queued files. The Queue Manager allows users to specify how, when, and where a file will be despooled and to display information about the queue. It controls the printing and provides all the services of the Serial Despooler, plus much more.

The Queue Manager handles the orderly printing of files for an RSX-11M-PLUS, RSX-11M, or Micro/RSX system through the despoolers, or print processors. There is a print processor for every lineprinter on a system. Even if a system does not have a hardware device of the lineprinter type, it will have some device designated as the system output device that takes the part of the lineprinter. This output includes all requests to system tasks for maps or listings, logs from batch jobs, and print jobs entered through the PRINT command, as well as output from applications tasks unique to an installation.

A print job can specify the forms required, the number of copies, the print priority, holding an entry, deleting files after printing, and printing after a specified time.

The print job can contain one or more files to be printed. Print jobs can be submitted by an interactive user, a program and, on RSX-11M-PLUS and Micro/RSX, by a batch job. Print jobs are automatically submitted at the end of a batch job.

Each print job has a large, easily read job header and file header burst pages, to identify print requests and files within a print request. Both types of headers contain identification and general accounting information.

THE PRINT COMMAND - The PRINT command spools print jobs and places them in a queue controlled by the Queue Manager for despooling. The most common use of this command is printing files on the system's lineprinter. Switches on the PRINT command can specify many attributes of the print job including

- The time when the spooling is to be done.
- The device that is to accept spooled output.
- The queue priority of the job.
- The restartability of the job.
- The forms the job is to be printed on.
- The number of lines per page.
- The number of copies of each file that are to be printed.
- Whether the job should be deleted after spooling.

The Queue Manager maintains, on disk, the queue of jobs to be printed. If the system is shut down, or if it crashes, none of the files that are yet to be printed are lost.

Print spooling under RSX-11M-PLUS and Micro/RSX has the additional capability of transparent spooling, wherein output to the lineprinter is transparently written to disk. When the file is closed, it is automatically queued to the lineprinter.

The following example illustrates the difference between spooling a file on RSX-11M and the transparent spooling on RSX-11M-PLUS. The commands are MCR commands.

On RSX-11M-PLUS

```
>DMP LP:=FILE.TMP
```

On RSX-11M

```
>DMP FILE.LST/SP=FILE.TMP
>PIP FILE.LST;*/DE
```

BATCH PROCESSING - RSX-11M-PLUS and Micro/RSX have a multistream batch processing capability that lets operations personnel control the number of batch streams that can run. Batch jobs can be submitted by an interactive user, a program, or another batch job. When the number of batch jobs submitted exceeds the number of streams, the remainder of the batch jobs are held in a batch queue. As with the spool queues, the operator can control the batch job queue by changing job priority, holding a job, or killing a job.

Volume mount commands issued in a batch job can request a generic device, such as a disk, or specific device unit, such as disk-drive unit 2. The batch job waits until the operator satisfies the mount request, while other batch jobs proceed.

THE SERIAL DESPOOLER TASK - The Serial Despooler Task provides a more primitive means of eliminating contention for the system lineprinter. Rather than waiting for the lineprinter to become available, a task directs the output intended for the lineprinter to a disk file. The task issues a Send Data directive to the serial despooler, placing a data block that identifies the file to be spooled in the serial despooler queue. A request directive is then issued by the task to activate the serial despooler in case it is not already active. The serial despooler handles FCS-created files, but RMS files can be read only if they are sequential. All files identified in the serial despooler queue are printed in first-in/first-out (FIFO) order.

- PROGRAMMING UTILITIES

RSX operating systems support two programming utilities — the Librarian Utility Program and the File Dump Utility — that allow users to work with library files and to examine file contents.

LIBRARIAN UTILITY PROGRAM - The Librarian Utility Program enables users to create, update, modify, list, and maintain object, macro, and universal library files. Library files, direct access files that usually contain modules of the same type, are organized for rapid access by the Task Builder, MACRO-11 assembler, and the system library routine.

The Librarian is invoked interactively via DCL or MCR commands. Once the Librarian is invoked, users can work with it directly or by means of indirect command files. This provides users with fast entry-point search time, easy update with minimal copying of entire files, and the ability to handle multiple module types.

There are two library types as defined below — macro libraries and object libraries. Digital makes system directives and system-related features available through calls. Definitions for the calls reside in macro libraries. The libraries are stored in a predefined file area known as the User File Directory (UFD). The UFD is located on the system library device.

To use these libraries, a user supplies in the source code the appropriate names of the modules as parameters of a MACRO-11 directive. This action tells the assembler to generate an entry for that call in its macro-symbol table and to search the appropriate library for the definition of the macro symbol.

These libraries provide the code that enables a task to issue system directives and to obtain File Control Services (FCS); allows software to refer to offsets for the executive data structures; and provides the definitions for Record Management Services (RMS) calls for sequential and relative file I/O.

On RSX-11M-PLUS, RSX-11M, and Micro/RSX, system object libraries provide general utility functions and special purpose executive features. These libraries, like the macro libraries, reside in the UFD on the system library device.

System library routines reduce program development time and decrease the use of memory by making routines that perform the following functions available to all users:

- Save and restore register contents for control transfer to subroutines.
- Perform integer and double-precision multiplication and division.
- Convert ASCII input data to binary and Radix-50 format, and vice versa.
- Convert and format output data to produce text for a readable printout or display.
- Manage memory and disk-file storage, to accommodate tasks that require large amounts of memory for data that must be transferred between memory and a disk work file.
- Obtain a command line from a terminal, indirect command file, or an online storage medium.
- Separate a command line into whatever appropriate data set descriptions are required by the file system for opening a file.
- Separate a user-defined command line with user-defined command syntax that includes built-in variables.

FILE DUMP UTILITY - The File Dump Utility (DMP) program produces a printed listing of the contents of a file or volume. The listing can be directed to any suitable output device—lineprinter, terminal, magnetic tape, DECtape, or disk. DMP runs in two modes—file mode and device mode.

In file mode, the user specifies one input file, and DMP dumps all or a range of virtual blocks of the named file. A virtual block refers to a block of data in a file. Any Files-11-structured volume serves as the input device for DMP.

In device mode, only the input device is specified, and a specified range of logical blocks is dumped. A logical block refers to an actual block on disk and DECtape, and physical records on magnetic tape and cassette.

- PROGRAM MAINTENANCE UTILITIES

Program maintenance includes modifying, patching, and comparing files. The four program maintenance utilities are the File Compare Utility, Source Language Input Program, Object Module Patch Program, and the Task/File Patch Program.

FILE COMPARE UTILITY - The File Compare Utility (CMP) compares two ASCII text files, line by line, to determine whether parallel records are identical. Using CMP, a user can

- Generate a listing showing the differences between the two files. Each difference is listed as a pair; first, the lines from the first file that are being compared to lines in the second file, then the lines from the second file are being compared to the lines from the first file.
- Generate a listing in the form of one list, with differences marked by change bars.
- Generate output suitable for input to the Source Language Input Program (SLP) utility (described below). This output contains the SLP commands and the input that is required to make the first input file identical to the second input file.

SOURCE LANGUAGE INPUT PROGRAM - The Source Language Input Program (SLP) is used for source-file maintenance. SLP maintenance is usually performed on the most recent version of the source file, ensuring that the file contains the latest updates and corrections. An optional audit trail in the output file allows a user to keep a record of changes to the software.

With SLP, a user can

- Update (delete, replace, add) lines in the existing file.
- Create source files.
- Run indirect command files containing SLP edit commands.

Both an input file to be updated and command input that consists of text lines and edit command lines specifying the update operations to be performed are input to SLP. To locate lines to be changed, SLP uses locators that are specified as line numbers or character strings. Command input can come directly from a user's terminal or from an indirect command file that contains commands and input lines that are to be inserted into the file. SLP accepts data from any RSX-11M-PLUS or RSX-11M file-structured device.

SLP output is a listing file and an updated input file. SLP provides an optional audit trail that helps keep track of the update status of each line in the file. Unless suppressed, the audit trail is shown in the listing and is permanently applied to the output file.

OBJECT MODULE PATCH PROGRAM - The Object Module Patch Program (PAT) allows a user to update or patch code in a relocatable object module. Input to PAT consists of two files—an input file and a correction file. The input file consists of one or more concatenated object modules that are connected individually by PAT. The correction file consists of object code that, when linked by the Task Builder, either overlays or is appended to the input object module. Unlike the Task Builder and patching options (described below), PAT allows users to increase the size of the object module because the changes are applied before the module is linked to the Task Builder.

PAT uses corrections and/or additional instructions in the correction file to update the object module. Correction input is prepared in source form and then assembled by the MACRO-11 assembler. Output from PAT is the updated input file.

TASK/FILE PATCH PROGRAM - With the Task/File Patch Program (ZAP), users can directly examine and modify files on a Files-11 volume. Users can patch data files or task images interactively without having to reassemble and rebuild the task.

ZAP performs many of the functions performed by the RSX-11 Online Debugging Tool (ODT), including

- *Command-line switches* that allow access to specific words and bytes in a file, modify locations in a task image, list the disk-block and address boundaries for each overlay segment in a task disk image, and open a file in read-only mode.
- A set of *internal registers* that includes eight Relocation Registers.
- *Single-character commands* that, in combination with other command-line elements, allow users to display, open, close, and manipulate the values in task images and data files.

Except in read-only mode, the results of ZAP commands are permanent. By using ZAP commands on a hardcopy terminal, users are assured a record of changes made during the patching process.

Using maps generated by the Task Builder, as well as listings generated by MACRO-11, users have sufficient information to make the required patches rapidly.

Building and Debugging Tasks

The basic unit of work that RSX facilities service is called a task. A task consists of a program written in a source language that has been assembled or compiled into an object format and then built into a task image by the link utility called the Task Builder.

▪ THE TASK BUILDER

The Task Builder is a multiple-purpose tool. It allows users to create a loadable entity (called a task image), define and structure a shared area of memory (called a resident common), and arrange shareable routines (called resident libraries) to reside in memory.

In addition to the normal linkage functions of combining object modules or creating overlays, the Task Builder sets up the basic task attributes that determine the task's resource requirements and relationships to other tasks in the system. The significant task attributes that affect a task's operation in a realtime multiprogramming environment are

- Partition — the section of memory in which the task will reside when it executes.
- Priority — the task's relationship to other tasks in competing for system resources.
- Checkpointability — whether or not a task can be swapped out of memory when it is not executing in order to make room for a task of higher priority that is ready to run.

Consistent with RSX system's flexibility goals, all these attributes can be changed when the task is installed or when it executes.

To build a task image, the Task Builder accepts, as basic input, the output of a language processor in the form of an object module or multiple object modules. The user directs the Task Builder to generate a file of executable code (the task image), a file of memory allocation information (a map), and/or a special file of symbol definitions used in constructing the task (the symbol definition file).

The task image, residing on a disk, is in a format suitable to be loaded into memory and executed. In addition, the Task Builder can generate a cross-reference listing showing all the global references.

In creating a task image, the Task Builder's primary functions are linking, address binding, and building system data structures. Linking involves resolving global references and program-section references for all object modules. Address binding is the assigning of virtual address space within the task. Building system data structures involves the creation of elements that the system needs in order to load the task image into memory and to execute it. To resolve global symbols that are not defined in any of the input object modules, the Task Builder searches whatever object libraries are specified and, as a default condition, searches the system object library.

- DEBUGGING AIDS
Debugging aids include an Online Debugging Tool (ODT), a Postmortem Dump (PMD), and a Snapshot Dump ($SNAP) facility to assist in identifying faulty code in a program.

ONLINE DEBUGGING TOOL - ODT allows interactive control of task execution. Programmers specify to the Task Builder that they want a debugging aid included in a task.

When a task that includes ODT is run, execution begins at the ODT transfer address, rather than at the task starting address. Therefore, ODT gains control and allows users to type special commands that establish base addresses and that set breakpoint locations within the task. After users tell ODT to begin task execution, ODT saves the instructions at the breakpoint locations specified and replaces them with PDP-11 breakpoint (BPT) instructions. ODT enables the SST (Synchronous System Trap) entry point in the task. Upon encountering a BPT instruction in the task, the executive passes control to ODT at its breakpoint routine. ODT saves task registers in special locations, restores instructions to the breakpoint locations, and transfers control to the user's terminal. By typing ODT commands, users can examine and alter any instructions or data within task memory.

If a task generates an SST error, ODT gains control at its SST entry point, prints a notice at the user terminal, and passes control to the terminal. The ODT commands can be used to discover the cause of the error, correct it, and perhaps continue executing the task.

POSTMORTEM DUMP - PMD is directed by the executive to extract runtime-related data pertaining to a terminated task, to format it, and to request a printed listing. PMD requires that the executive option for abnormal task termination and device-not-ready messages be selected at system generation. Normally, when a task generates an SST, such as what would be caused by an improper reference to an odd address or a reference to a nonexistent memory location, the executive tries to transfer control to an SST entry point defined by the task. If the task does not have an SST routine defined for the particular type of trap, the executive begins task termination.

By enabling Postmortem Dumps for a task that does not handle SSTs, users tell the executive to supply more data at abnormal task termination. In other words, the executive follows the abort procedure and, in addition, creates a request for PMD to create the dump. PMD examines system and task structures to preserve status and runtime data, reads the task image from memory, and writes it to disk in a readable format. PMD then queues a request to print the file containing the dump data, after which the executive completes the task abort procedure.

SNAPSHOT DUMP - $SNAP generates an edited dump of a running task. The snapshot dump requires users to insert special code in a task. Although more complex than PMD, $SNAP allows users to choose the location at which the dump is created and to select the extent and format of the dump. In addition, users can generate the dump from more than one location and can, therefore, generate as many as needed during task execution.

It is often useful to include such debugging facilities as $SNAP in a task based on defining a conditional variable. The facility can be included, while debugging, by defining the variable. The facility can then be omitted by reassembling the code with the conditional variable undefined.

▪ RSX-11S and Other Special Considerations

RSX-11S requires an RSX-11M-PLUS, RSX-11M, or VAX/VMS system for system generation and program development. An RSX-11S system is generated by a process very similar to the RSX-11M system generation process. The maximum hardware configuration is the same as that of an RSX-11M system.

Because it is based on RSX-11M, RSX-11S enjoys most of the inherent features and generation capability of that system. For example, RSX-11S automatically supports all of the peripheral devices that RSX-11M supports, as well as such hardware features as Floating-Point Units, parity memory, and memory management. All are selectable at system generation. When included in an RSX-11S system, these features, of course, take up additional memory.

However, because RSX-11S is memory-resident, it does not support a file system, nonresident tasks, checkpointing, disk-resident overlaid tasks (memory-resident overlaid tasks are supported), or program development. Its main purpose is to provide a runtime environment for executing tasks.

The basic software building blocks of an RSX-11S system are a subset of the features available in the RSX-11M executive and a special File Control Services (FCS) that contains no support for directory devices.

The minimum software system is one with only an executive. The smallest executive that can be generated requires 5 Kbytes of memory. However, the executive must be augmented with other software for it to do useful work.

The following services, omitted from the minimum 5-Kbyte executive, can be generated into an RSX-11S system and take up additional memory:

- Address checking.
- Asynchronous System Traps (required for FORTRAN).
- I/O rundown.
- External MCR functions (user-written functions).
- Install, request, and remove-on-exit support.
- SEND, RECEIVE, GET TASK PARAMETERS, GET SENSE SWITCHES, and GET PARTITION PARAMETERS directives.
- Parity memory support.
- Network support.

In addition, the following programs can be included in an RSX-11S system:

- All RSX-11M I/O device drivers, except the console-logger device driver.
- Basic MCR.
- Online Task Loader.
- Task Termination Notification program.
- System activity display programs.
- SETTIM subroutine used to set the system's internal time.
- System Image Preservation program.

Device Drivers

If the PDP-11 system includes I/O devices, the RSX-11S software system must include the executive and the appropriate I/O device drivers. All RSX-11M I/O device drivers, except the console-logger device driver, are supported.

Most drivers use an average of 3 Kbytes; for example, the lineprinter driver needs less, the terminal driver more.

Basic MCR

If operator communication is required, Basic Monitor Console Routine (MCR) can be included in a system. It takes up an additional 6 Kbytes of memory. Basic MCR provides commands to control and modify the execution of tasks installed in an RSX-11S system. It is a subset of the RSX-11M MCR. Both Basic MCR and MCR are strictly subset compatible.

Online Task Loader

The Online Task Loader (OTL) can be included in an RSX-11S system if online loading of tasks is desired.

Tasks are created on a host RSX-11M-PLUS, RSX-11M, Micro/RSX, or VAX/VMS system, transferred to the load medium using the File Exchange Utility (FLX), and then loaded into a running RSX-11S system using OTL. OTL expects the load-device medium to be written in either DOS or RT-11 format, depending on the load device.

OTL reads a task image from the load-device medium, verifies the task-image format, and checks for error conditions. It then creates a Task Control Block (TCB) for the task, places the TCB in the System Task Directory (STD), and fixes the task in memory, making the task known to the system. For system-controlled partitions, OTL allocates a subpartition PCB (Partition Control Block) for the task.

The minimum size for OTL is 7 Kbytes. In 7 Kbytes, however, OTL supports only one load device. Online taskloading requires a 32-Kbyte system, because approximately 21 Kbytes will be required for system software (5 Kbytes for the executive, 6 Kbytes for Basic MCR, 3 Kbytes for one device driver, and 7 Kbytes for the OTL).

Task Termination Notification Program

The Task Termination Notification program (TKTN) outputs abnormal task termination and device status information.

If a task aborts, TKTN prints on the task's initiating terminal the notification and appropriate error message. TKTN also displays the contents of the task's registers at the time the task aborted. If a task aborts with I/O requests outstanding, the error message will include the phrase "and with pending I/O requests."

All task-abort messages are printed or displayed at the initiating terminal. All other TKTN message are printed on the console terminal. TKTN requires an additional 2 Kbytes of memory.

System Activity Display Programs

System activity display programs are a set of privileged tasks that display information concerning task activity in an RSX-11S system. Users can invoke and continually run these tasks on Digital's VT200 and VT100 videoterminals. The display format is similar to that of the RSX-11M-PLUS and RSX-11M RMDEMO display. These programs require approximately 12 Kbytes of memory.

The programs display the following information:

- Current date and time.
- The currently active task.
- All tasks, loaded drivers, and common blocks currently in memory, displayed to show their individual memory requirements and locations relative to other tasks.
- The number of active tasks currently in memory and the total amount of memory occupied by each task.
- The current amount of available system dynamic memory (pool), including the largest available block, the number of fragments, and the worst case of pool space since bringing up the display programs (not displayed on a VT05-B).
- A graphic display of partition information.

SETTIM

SETTIM, a FORTRAN-callable subroutine, is used to set the system's internal time. It is supplied to allow a running program to set the time in a configuration that does not include a console terminal or Basic MCR. The module is included in a user task by linking with the file SETTIM.OBJ.

System Image Preservation

The System Image Preservation (SIP) program is an online utility task that can save the image of a running system on a load device medium in bootstrap format. The saved system can subsequently be restored by bootstrapping it from the load device medium. The minimum size for SIP is 3 Kbytes. In 3 Kbytes, it supports only one load device.

File Control Services

The standard RSX-11M File Control Services (FCS) record I/O package contains a large amount of code to support file-structured devices. Because RSX-11S contains no file support, this code is unnecessary. The special version of FCS provided with RSX-11S is the standard FCS without the file support code. This provides a significant size reduction.

The RSX-11S FCS subset does not support

- File-structured devices (Files-11 disk and ANSI magnetic tape).
- Block I/O operations.
- Random access record I/O.
- Sequenced records.

In general, the subset FCS handles I/O in the same way as the RSX-11M FCS handles I/O for record I/O operations.

Virtual MCR

The Virtual Monitor Console Routine (VMR), an RSX-11M-PLUS, RSX-11M, and Micro/RSX system program, allows the complete interactive configuration of an RSX-11S system on an RSX-11M-PLUS, RSX-11M, Micro/RSX, or VAX/VMS system. Users can define partitions, load and unload device drivers, install and fix tasks, and establish other system parameters. Users can then transfer the entire system to an appropriate load medium for bootstrapping on the target RSX-11S system.

RSX-11M-PLUS, RSX-11M, and VAX/VMS use the VMR for generating an RSX-11S system.

▪ VAX-11 RSX

VAX-11 RSX is the optional software product that allows RSX-11 task images to execute in the VAX/VMS environment and makes it possible for task images from an RSX-11 operating system to migrate to a VAX/VMS system and to execute there. For more detailed information about VAX-11 RSX, refer to Chapter 24, "Cross-system Coexistence and Migration Tools."

▪ A-to-Z Integrated System Software

A-to-Z Integrated System software was specifically designed to provide solutions for small-business, vertical market applications, by integrating functions such as word processing, graphics, report writing, spreadsheets, mail, and accounting.

For a discussion of these integrated applications, refer to Chapter 23, "A-to-Z Software."

Chapter 4 ▪ RSTS Operating System Family: RSTS/E and Micro/RSTS

▪ RSTS Is Designed to Build and Run Commercial Applications

Thousands of users, from financial institutions and schools to manufacturers, insurance companies, and airlines, find RSTS/E to be a system that meets the computing needs important to the commercial and administrative environment — reliablity, security, consistency, low cost-per-user, and an efficient base for building and running commercially oriented applications.

These features, in the course of over a decade, have led to the development of a wealth of applications suited to all dimensions of the commercial and administrative marketplaces. Applications range from general use products such as word processing, spreadsheets, inventory control, and accounting to specialty applications such as golf handicapping, chromatography graphics, and legal analysis.

RSTS/E allows concurrent word processing and data processing using DECword/DP and can also communicate with stand-alone word processors. Its built-in and layered functions reflect Digital's commitment to keep the system easy for naive users and yet extremely powerful for users who want to write complex or innovative programs.

Excellent communications software available for RSTS/E lets the computer link into distributed networks of Digital computers (DECnets) or into flexible Internets with computers from other manufacturers. Also, thanks to the availability of a DCL subset with RSTS/E, there is increased compatibility among RSTS/E, VMS, and RSX operating systems.

System accounting facilities included with RSTS/E give the system manager a detailed record of who used the various processor modes and for how long, so that both system management and billing for timesharing can be done accurately.

There is an enormous amount of specialized software available for customers, from Digital and from commercial developers who specialize in writing program packages for RSTS/E users in numerous industries. Resources of this sort help every customer who needs to enhance the operating system with software designed for financial accounting and general ledger, billing, forecasting, business simulation, materials control, a variety of banking transactions, freight tracking, insurance claim processing, and hundreds of additional timesharing applications. Digital's PDP-11 Software Source Book (see Appendix C) describes over 750 application products available to users of the RSTS Operating System.

The RSTS family includes two members — RSTS/E and Micro/RSTS. RSTS/E is designed for people who need the maximum flexibility in system configuration, a complete powerful program development environment, and a full range of user, management, and program development documentation.

RSTS/E and Micro/RSTS systems include the MACRO-11 assembly language and such optional, high-level languages as BASIC-PLUS, BASIC-PLUS-2, COBOL-81, FORTRAN IV, and FORTRAN-77. DIBOL, another high-level language, is supported by RSTS/E only.

Micro/RSTS is designed for the MicroPDP-11 family of computers and meets the needs of people whose primary use of the system will be running RSTS-based applications. Almost all applications that run on RSTS/E and the MicroPDP-11 hardware will also run unmodified on Micro/RSTS. Micro/RSTS includes the BASIC-PLUS language and is offered at a lower cost than RSTS/E but does not include many of the powerful development tools and the configuration flexibility that are standard features of RSTS/E.

Features of both RSTS/E and Micro/RSTS systems are listed in the pages that follow. All have as their goal the creation of a highly "available," programming environment — an environment in which programmer productivity is maximized, downtime is reduced to a minimum, and system operations are easy to learn and manage. RSTS/E provides excellent security, particularly useful in sensitive business applications such as bank transactions and stock transfers, in which access to certain data must be severely restricted; or for educational institutions to prevent downtime resulting from inexperienced or unprivileged users.

▪ Interactive Timesharing Provides Fast Response Time

Because computer hardware can really do processing of only one program or task at a time, it is important to determine, based upon the uses to which a computer is to be put, how that central processor is to be allocated. In timesharing environments, the processor is scheduled — usually in a round robin fashion — among all the jobs that want it and are ready to do useful work. There may be levels of priority in timesharing so that, for example, agents who are confirming hotel reservations get served ahead of clerks who are doing inventory control programs; but every timesharing system accounts for all executable programs eventually.

Because RSTS/E is a timesharing system, each user can be given equal access to system resources. RSTS/E allows good terminal response time for a large number of concurrent users. Many tools are available to the system manager to control and optimize the system's usage. From the user's point of view, the most important timesharing parameter to consider is computer response time—the time that elapses between entering an instruction or field of data and the computer's response computation, a ledger entry, or an output operation to a printer or terminal.

RSTS/E supports as many as 63 concurrent users (jobs). Applications can be written to control multiple terminals. Often as many as 127 terminals can be active at the same time. Micro/RSTS, optimized for the MicroPDP-11 environment, can support as many as 14 terminals and is extendable to 20 jobs. The additional jobs can be used to do tasks that do not require terminal interaction without decreasing the number of available terminals.

RSTS/E systems provide excellent response time to users, who can number up to 127 at one time, and to jobs that can number up to 63 at one time. A typical mix of users might include some people doing program development and working in a very interactive mode with the computer, some clerks doing data entry for delivery schedules or invoicing, some salespeople entering transaction information, and perhaps even a batch job or two being run to complete the weekly payroll. In addition, another computer owned by the company might be providing delivery status information via DECnet, or a mainframe from another computer manufacturer might be linked through Internet software from Digital.

With such a diverse mix of users, you might find some people who know very little of the internal working of the computer, some who are programmers and expert in a language such as COBOL or BASIC-PLUS-2, and some who are system-level programmers and system managers, capable of adapting the software to a variety of specific needs and of tuning the operating system for maximum performance. Under RSTS/E, all will get excellent response time, and all will feel as though they have unique control of the central processor and other resources of the computer.

RSTS/E keeps the CPU busy by running several jobs concurrently. Each user program, be it a system utility, runtime system, or application program, is a job. A job runs until it either enters an I/O wait state or exhausts its time quantum. At that point, the scheduler finds the next ready job and begins running it. Meanwhile, the interrupt-driven I/O device handlers are processing requested data transfers. Upon completion of a transfer, the scheduler marks the job that requested the transfer as ready to run again and starts it from the point at which execution ceased.

RSTS/E keeps as many jobs in memory as possible. When more memory than is available is required to run a job, the system temporarily swaps some jobs out of memory and stores them in a swap file. When the job's turn to run again occurs, the job in the swap file is brought back into memory. Jobs waiting for more CPU time or keyboard input are most likely to be stored in the swap file. Jobs currently running or involved in disk or magnetic tape data transfers are necessarily in memory.

As the system processes each job, it maintains accounting information concerning that job. When the user logs off the system, all the information accumulated for all the jobs run by the user is used to update the accounting information stored on disk for that user account. This is particularly important to systems in which time is billed among various users.

To begin a timesharing session, a user logs into the system by entering an account name and a password (these are assigned by the system manager, the password is determined jointly by the system manager and the user). The terminal is then under control of the keyboard monitor of the system default runtime system. The recommended default runtime system is DCL.

Whatever the default runtime system is, after the log in verification is complete and the system messages have been displayed, the user is in command mode. Each runtime system identifies itself by an identifying prompt. These are

- DCL "$"
- RSX ">"
- BASIC "Ready"
- RT-11 "."

These commands, issued to the keyboard monitor of the runtime system, cause the execution of Commonly Used System Programs (CUSPs) or application programs. The user is permitted to execute all the commands available to a nonprivileged user. Privileged users have additional commands available for system management and maintenance.

A privileged user can detach the running job from the terminal, and run another job. The detached job continues to run unattended, but is still associated with the account number under which the user logged in. To retrieve control of a detached job, the user can log in on any free terminal and attach that job to the terminal.

The use of BASIC-PLUS is an important feature of the RSTS/E system. BASIC-PLUS can be run either from any default runtime system by issuing the BASIC command or it can be a runtime system itself. When BASIC-PLUS is entered, it is in edit mode, to which it returns when program execution is completed or whenever a CTRL/C is typed. In edit mode, BASIC-PLUS examines each line typed by the user and determines whether that line is

- A system or installation defined command.
- An immediate mode statement.
- A program statement.

System- and installation-defined commands are executed immediately after being entered; immediate mode statements are first translated into an intermediate code (placed in the user's job area) and are executed immediately by the runtime system. Program statements (lines of ASCII text preceded by the line numbers) are stored in their ASCII form in a temporary disk file under the user's account. Program statements without line numbers are immediately executed and not stored. This feature is provided for program debugging.

A user's job area is initialized either by executing the BASIC command or by logging in and being given a size of 2 Kbytes or 4 Kbytes, depending on the runtime system being used. The job area can grow in increments of 2 Kbytes to a maximum size chosen by the system manager. When the user enters program statements in the edit mode, intermediate code created in the user's job area is not executed automatically. A copy of the intermediate code of the program can be transferred to disk storage or to an external storage medium.

You can change from edit mode to run mode by typing the RUN system command or the CHAIN immediate mode statement. In RUN mode, the runtime system interpretively executes the intermediate code stored in your job area. When a program finishes execution, the terminal returns to edit mode as signaled by the printing of the prompt "Ready." You can interrupt the BASIC-PLUS program by typing CTRL/C, which also returns the terminal to edit mode.

EDT is the standard editor offered on Micro/RSTS and RSTS/E operating systems. It is easy to learn, with editing instructions consisting of English words or their shortest unique abbreviations. Extensive HELP facilities remind you quickly of the possible options for a particular command and the format for that editing instruction. EDT doesn't modify the input file directly, so that if a user accidentally deletes a large amount of text, the original input is still available for quick recovery. For a more detailed description, refer to Chapter 20, "EDT."

▪ Resources Provide the Services You Want from an Operating System

Because high programmer productivity was Digital's aim in producing RSTS/E, the features of an easy, forgiving environment are included with the system. User commands to the RSTS/E system are handled and interpreted by one of the runtime systems capable of acting as a keyboard monitor. The most popular languages for business and educational applications are available, including BASIC-PLUS, BASIC-PLUS-2, COBOL-81, FORTRAN-77, FORTRAN IV, and DIBOL. This choice of languages lets you adapt your language to the function at hand, and lets you tap programming talent already trained in these popular languages. If you are familiar with traditional batch-mode program development requirements, you will be impressed by the ease and speed of developing applications under a RSTS/E system.

Text editors, particularly the Digital standard editor (EDT), help speed program development and correction. EDT is a text editor that can be used to create a file, enter and manipulate text in the file, and save or delete work done during edit sessions. It is easy to use and, along with Digital's terminals, provides full screen video editing capabilities. In actual applications, the RSTS/E user can take advantage of such features as Record Management Services (RMS), software that supports building and accessing sequential, relative, and multikey indexed sequential file structures, and relieves programmers of many tedious tasks of I/O management. Calls from most of the various programming languages to RMS are sufficient to invoke the utility or access method desired.

FMS-11/RSTS Forms Management System is a software package that provides sophisticated screen formatting for application programs. FMS-11/RSTS allows nonprogrammers to design forms interactively on the videoscreen and eliminates tedious editing and recompiling of a forms program. The keypad-operated editor and the HELP facility are easy to learn. FMS-11/RSTS provides extensive field protection and validation features that help prevent data errors caused by typing errors.

▪ Dynamic Allocation of System Resources Lets All Users Receive a High Level of Service

RSTS/E users can expect efficient operation because the operating system dynamically allocates processor time, memory space, file space, and peripherals to best suit changing demands. The system manager and designated privileged users have access to the RSTS/E system management commands either interactively using system utilities or under program control. Additional system commands and utility programs are also available to all users.

The RSTS/E file system provides a wide range of online processing capabilities. Files can be accessed randomly or sequentially, either through one of the keyboard command or utility programs or through the RSTS/E file system. Files can contain alphanumeric string, integer numeric, floating-point numeric, or binary data; they can be created, updated, extended, or deleted interactively either from the user's terminal or under program control, and can be sorted by the SORT-11 program. Files can be protected from access on an individual, group, or system basis; they can also be accessed by many users while being updated online.

Total or selective file backup and restore can be done online without disrupting users, or it can be done during periods when timesharing or application processing is not permitted. Private disk volumes may be used to limit user access. Removable disk media permit safe storage of valuable records at sites remote from the system.

RSTS/E is a high-performance system that includes a variety of user tools to tune applications to perform even better. For example, with software disk caching (unrelated to CPU memory caching, which is also available), blocks of heavily used disk data are held in main memory to reduce the number of disk accesses. In addition, heavily used program segments can be held resident in main memory and shared among programs, saving memory space, reducing swapping time, and thus increasing performance. RMS data management code can also be shared for further memory savings. Use of common, shared, resident RMS also results in substantially reduced disk accessing for overlays and thus improves applications performance.

▪ RSTS Software Can Accommodate Additional Hardware As Needs Increase

RSTS/E runs on a wide range of Digital's processors, accommodating the complete range of peripherals, subsystems, and PDP-11 add-ons to meet the need of the commercial and administrative user servicing any market. RSTS/E is well adapted to the growth of an individual system — as hardware and software are added — and is fully upward- and downward-compatible with applications being migrated from the smallest MicroPDP-11 to the largest PDP-11/84 systems.

▪ Software Options Are Chosen during System Generation

System generation is normally a one-time operation in which the system manager defines the hardware configuration and selects the basic software options. The system manager needs to perform a system generation only when the system is first installed or when the hardware configuration changes. Software options can be included in the system to tailor the system to the needs of the application.

In addition to defining the number and kinds of peripherals and processing hardware during system generation, the system manager defines special configuration options. Some of these options are discussed below.

▪ System Software Extends and Enhances the Capabilities of the Hardware

RSTS/E system software exists as system code, language processing code, and system program code. The system code and language processing code are tailored at system generation time according to the hardware configuration on which the system runs and the software features chosen by the system manager. Once the system is generated, the system code and language processing code are frozen, and are alterable by patching or generating new code. The system program code exists in a library of programs executable by the system software or by individual users on the system. The library of programs is alterable and expandable during timesharing without requiring regeneration of the system.

System Code

The RSTS/E system code is stored on the system disk as a save-image library (SIL). A SIL, when loaded into memory, is immediately executable by the PDP-11 computer. The system code comprises many distinct elements that are either resident in memory or on disk during timesharing. Permanently resident elements are the following:

- Interrupt and trap vectors.
- Small and large system buffers.
- System information and data tables.
- Disk and device drivers.
- File processor modules.

The following are optionally resident modules:

- FMS/RSTS forms code in the terminal driver.
- DECnet/E—Network Communications handler.
- RJ2780—Remote Job Entry handler.

File processor modules and infrequently used utility routines are either permanently resident or disk resident (overlay) elements, and may be selected during system generation. System initialization code is loaded only during system startup.

RSTS/E operations start when the system disk is bootstrapped. The bootstrap routine loads the initialization code that determines the hardware configuration and performs many consistency checks to ensure the integrity of the software. When checking is completed, the initialization code remains resident and allows many options.

Systems Initialization

After generating the system, the system manager bootstraps the RSTS/E system to load the initialization (INIT) code into memory. The INIT code is a collection of routines used to create the file structures, system files, and startup conditions required for normal operation of the RSTS/E system. The INIT code is essentially one large stand-alone program with many functions. INIT includes routines that ensure the integrity of disk file structures and perform many checks on the hardware configuration. Options are provided that enable the system to function even when certain hardware elements are inoperative. Finally, the initialization code is responsible for loading the RSTS/E Monitor into memory for normal timesharing operations.

Once the default system initialization startup parameters are set up, the system manager does not have to repeat manual startup each time the system is started. Using the automatic restart feature, the RSTS/E system can recover and restart the timesharing session automatically after a system malfunction or power failure. When the system is started in automatic restart mode, control bypasses all parts of the startup code that call for operator intervention.

▪ System Function Calls Bypass the Runtime System

Many programs, particularly MACRO-11 assembly language programs, are required to perform functions that directly affect the way in which the system handles such things as memory, I/O devices, runtime systems, etc. Rather than allow the application or system programmer to perform these manipulations in an uncontrolled way, the RSTS/E system provides a series of System Function Calls. These are calls from the program directly to the monitor that bypass the runtime system. The monitor performs the function, and then returns to the calling program.

Some of the system function calls are privileged and can be issued only by privileged users or jobs while others are available to all users. This ensures that the unprivileged user cannot gain access to system functions that could drastically change the way the system is supposed to operate.

System function calls are available to most of the high-level languages available on RSTS/E. The calls are formatted for each language consistent with the general language syntax. One major fact is that the calls are very system-dependent.

The *PEEK* system call is a privileged system call that allows any privileged job to examine any location in the monitor part of memory. This allows the user to read the monitor code, monitor data structures, including data structures of other users, and the file processor. This function does not allow a user program to examine the contents of another user's program.

▪ Micro/RSTS Is an Execute-Only System Designed Especially for the Micro/PDP-11

Micro/RSTS, a compact subset of RSTS/E, is an application-only, timesharing system designed to run RSTS-based programs. It runs exclusively on the Micro/PDP-11 and typically serves one to four users in a small business or department.

Micro/RSTS disk requirements are small—about 2.5 Mbytes in contrast to 10 Mbytes for RSTS/E. Programs for Micro/RSTS are coded, compiled, linked, and debugged on the full RSTS/E system and then loaded into the smaller Micro/PDP-11 system for execution. Applications can be developed on one RSTS/E system for use on several scattered Micro/RSTS systems.

Micro/RSTS is a pregenerated system that you can install with an interactive installation procedure. There is low system management overhead because there are no SYSGEN requirements.

The following features describe a Micro/RSTS system.

- A maximum of nine user jobs can run on Micro/RSTS at one time. However, multiple-terminal service can accommodate up to 12 terminals.
- Any terminal can be connected to the system remotely through an appropriate modem.
- Directory caching allows the saving of file device directories in main memory for quick access.
- If the error handling code can't recover from an error, the read/write area of memory is dumped into a file named CRASH.SYS on the system disk.
- Echo mode for terminal I/O allows typed characters to be taken away from the current cursor position and echoed elsewhere on the screen.
- As many as 50 systemwide logical names can be given to devices.
- Shareable resident libraries of routines or data can be included on the system disk and clustered to save memory.
- The Forms Management System (FMS) is supported.

Runtime Systems

Micro/RSTS includes three runtime systems. The *DCL* (Digital Command Language) runtime system's keyboard monitor is the default interface between the user and Micro/RSTS. It recognizes a subset of the RSTS/E DCL commands. The *RSX* runtime system provides a versatile programming environment compatible with the VAX/VMS operating system. The *RT*-11 runtime system (with its limited range of languages) is best suited to run existing RT-11 programs, rather than to develop new ones.

Micro/RSTS Utilities

Micro/RSTS supplies many RSTS/E utilities including

- SORT—sorts a selected key field in each record of a file in either ASCII or EBCDIC collating sequence. The DCL Sort commands are included in the base system. The Application Development Kit is required for the callable interface to user programs.
- EDT—the standard text editor. EDT allows users to recover editing work performed prior to a system crash.
- BACKUP—backs up disk data by copying selected files to other media.

Application Development Kit

The Micro/RSTS Application Development Kit provides software and documentation in support of native MACRO-11 assembly level language and high-level language program development. The Application Development Kit includes

- MACRO-11 programming — with an assembler, a FILE DUMP utility, a Resource Monitor Display (RMD), an Online Debugging Technique (ODT), and MACRO libraries.
- System programming tools — with the necessary RSX components to develop privileged software such as user-written device drivers. It also includes tools such as a loadable executive debugger and loadable crash dump drivers to assist programmers in developing privileged code.
- RMS (Records Management System) backup and restore utilities.

Command Languages Let the System Perform Predefined Operations

A *command language* is the vocabulary used by a program or set of programs that direct the computer system to perform predefined operations. RSTS/E uses three such command languages, the *User Command Language*, the *Digital Command Language*, and the *Concise Command Language*.

The command language interpreter is interactive, comprehensive, easy to use, and very flexible. It enables the user to log into the system, manipulate files, develop and test programs, and obtain system information.

The four standard keyboard monitors are DCL, BASIC-PLUS, RSX, and RT-11. All of these interpret sets of system commands, that is, words followed by optional command parameters. These system commands allow users to perform all the fundamental functions required to use the RSTS/E system, such as logging on and off, and running programs.

Digital Command Language (DCL)

DCL is based upon the language available on most Digital operating system. In addition to fundamental operations such as listing directories and copying files, DCL includes features such as user-defined command synonyms, string and numeric symbol substitution, reading and writing files, system and account management functions, security control such as setting passwords, submitting to batch, setting terminal characteristics, and terminal activity logging.

A comprehensive set of HELP frames explaining the operation of all DCL commands is available to users at all times through the use of the "HELP" command. Entering the HELP command alone will provide a directory of subjects and descriptive information on the specific topic. Often, additional subtopics provide more specialized information that can then be selected by the user. All HELP information is provided in the Base Kit; however, installation of the HELP information is optional.

DCL command file processing provides the DCL user with the capability of placing a collection of DCL commands in a file and executing the commands as if they were a program. Conceptually, the commands in the file appear to the system as a series of commands that are entered at the keyboard. All command file processing is done within DCL so that no additional keyboards or job slots are required. Information can be passed to the command file processor at the time the file is invoked and used to customize the action taken as the command file is executed.

In addition to the standard DCL commands, the command file processor interprets a set of specialized commands that allow operations such as conditional branching, local and global symbols, special purpose DCL functions, error handling, "Control C" trapping, up to 13 nesting levels, selective display, optional time stamping, and user prompting.

DCL command files are used to start up the system and automatically set up system, group, and individual user environments when the user logs onto the system.

Concise Command Language (CCL)

The RSTS/E system commands issued by the user at a terminal are familiar words or abbreviations. The system accepts both long and short command formats for inexperienced and experienced users. It responds with understandable statements and, if a command does not supply complete information, prompts the user for remaining data. CCL commands allow you to enter one command that runs a system utility and specifies a single command for the utility to execute. The number of CCL commands that can be defined varies from system to system, depending on the number of "small buffers" configured into the system. An average system probably includes a fairly standard set of CCL commands for certain RSTS/E utility programs. The system manager has the option of freely adding to, deleting from, or modifying the standard set of CCL commands.

▪ Logical Disk Structures Let You Access System and User Data As Well As Executable Code

Logical disk structure is divided into two types — public and private. The file structure on a disk, whether public or private, is the same. On a public disk, any user can create files. Every user has an account on a public disk. There is always at least one public disk on the system, which is called the "system disk." All public disks together on a system are called the "public structure" because the system itself treats all the public disks together as a unit. For example, when a program creates a file in the public structure, that file is placed on the public disk that has most space available. This is done to ensure proper distribution of files across the disks in the public structure.

The system disk contains the system code. Language processors and the library of system programs are also contained on the public structure. Storage of active user jobs that are temporarily swapped out of memory are in swapping files, at least one of which is on the system disk.

Any remaining disk drives in the RSTS/E disk structure can be devoted to private disk packs or disk cartridges. A private disk is one that belongs to a few user accounts, conceivably to a single user account. Files can be created only under these accounts, and can be read (or written) by other users only if the protection code of the file permits. A user who does not have an account on a private disk cannot create a file on it.

▪ You Have a Choice of Three File Access Methods

The file access methods available for the RSTS/E system include

- RMS-11.
- BASIC-PLUS.
- DMS-500.

RMS-11

RMS-11 is the main file and record access method available on RSTS/E. It is used by BASIC-PLUS-2, COBOL-81, FORTRAN-77, and optionally on MACRO-11 and DIBOL-11. In addition, most of the utility programs and layered software products, e.g., SORT-11 and DATATRIEVE-11, will work only when using files maintained through RMS-11.

RMS-11 supports three file organizations — sequential, relative, and indexed. The indexed file organization allows each indexed file to have one primary key and up to 254 alternative keys. In addition to random access based on key values, programs can access records in an indexed file sequentially in ascending order by key values. Records are stored physically only in primary key order.

RMS-11 supports four record formats:

- Fixed-length records.
- Variable-length records.
- Variable-length records with fixed control fields.
- Stream records.

Indexed files are restricted to either of the two record formats—fixed or variable. The stream record format is restricted to sequential disk files only. Languages that do not use RMS (e.g., FORTRAN IV) cannot process RMS files unless the record format is stream.

User programs are provided with logical data record access to RMS files through extended language syntax statements. The form of the statements depends on the application language interface. The functions provided are

- OPEN.
- CLOSE.
- READ/GET.
- WRITE/PUT.
- REWRITE/UPDATE.
- DELETE.

In addition to the facilities provided for programming languages, there is a set of RMS-11 utilities that enable the user to create, load, maintain, and backup RMS-11 files.

BASIC-PLUS

BASIC-PLUS on RSTS/E provides three methods of file access:

Formatted ASCII	For standard sequential I/O operations.
Virtual Arrays	For random access of large data files. A virtual array is stored on disk and can contain string, integer, and floating-point matrices.
Record I/O	Allows the user to have complete control over I/O operations.

Formatted ASCII data files are the simplest method of data storage, involving a logical extension of the BASIC-PLUS PRINT and INPUT statements. The INPUT statement allows data to be entered to a running program from an external device, for example, the user's keyboard, a disk, DECtape, or paper tape reader. The PRINT statement causes the output of a specified string of characters to a selected device.

The PRINT-USING statement allows the user to control output formatting. A special set of formatting characters allows the user to format strings and numeric fields with tabs, special characters, and punctuation. For example, the user can format check amounts with asterisk-fill for protection.

The RSTS/E virtual array facility provides the means for a program to operate on data structures that require fast random access processing yet are too large to be accommodated in memory at one time. To accomplish this, RSTS/E uses the disk file system for storage of data arrays, and maintains only portions of these files in memory at any given time.

Virtual arrays are stored as unformatted binary data. This means that no I/O conversions need be performed in storing or retrieving elements in virtual storage. Thus, there is no loss of precision in these arrays, and no time wasted performing conversions.

The third type of I/O, record I/O, permits a program to have complete control of I/O operations. Record I/O is a flexible and efficient technique of data transfer. Input and output to record I/O file is performed by the BASIC-PLUS GET and PUT statements. These statements allow the user to read or write specific blocks (physical records) of a file, where the block size depends on the type of device being accessed. For example, disk file blocks are always 512 bytes long, while records from a keyboard device are one line long, where a line is delimited by a carriage return or similar terminating character. With disk files, the program has the capability of performing random access I/O to any block of the file. Furthermore, using record I/O operations, the user can create a logical organization for file formats by controlling record length.

Pseudo Keyboards

The system manager can define the system to have one or more pseudo keyboards. Using a pseudo keyboard as a communications device, you can write a program to control other jobs. In addition, each copy of the BATCH system program requires one pseudo keyboard to run jobs in a batch stream.

Multiple Terminal Service

The multiple terminal service option allows one program to interact with several users simultaneously by servicing their terminals on one I/O channel. This eliminates the need to run separate copies of the same program when several terminals must perform a similar function.

Floating-point Precision and Scaled Arithmetic

The system manager can select either single-precision (2-word) or double-precision (4-word) floating-point numeric format. If the system has floating-point hardware, the system manager can select a floating-point math package that will increase processing speed by using the hardware instructions. The scaled arithmetic feature is included in all 4-word floating-point math packages. Scaled arithmetic avoids loss of precision in floating-point calculations; it is therefore very useful in calculating sums of money that cannot be manipulated easily as integer quantities.

Systemwide Logical Names

RSTS/E allows the system manager to assign as many as 50 logical names on a systemwide basis. Any user can type a systemwide logical name to access the device (and, optionally, the account) it represents.

System Accounts and Libraries

RSTS/E systems have three system accounts integral to the operation of the system and have auxiliary accounts for more efficient operation of the system. The Master File Directory (MFD) account is used on the system device and other disk devices in the system to control system access. The system library account is used by the RSTS/E system to manage a library of generally available and restricted use system programs and message and control files. A third special system account contains the RSTS/E monitor files and routines that are critical to the operation of the system.

Of particular interest to the system manager is the accounting information maintained on each user account in the MFD on the system device. This accounting information is normally accessed through the system accounting utility programs. The system manager or privileged users can also access and change this information for programmers using the SYS monitor functions.

Language Processing Code

DCL serves as the recommended default runtime system. However, any of the languages mentioned above may be used for applications programs. The auxiliary runtime system or object time system associated with a given language processor is loaded into memory only when a request is made to execute that language compiler or to execute a compiled program written in that language. The language processors reside on the system disk in machine executable form and can be either permanently resident in memory or temporarily resident (swappable). Usually the language compiler is swapped out to disk as required, just as any normal user job would be.

The runtime system may vary in size from 4 Kbytes to 32 Kbytes, and is generally shared among users.

System Program Code

A library of programs is produced and stored on disk during the system library build procedures of system generation. Both the system and users execute these programs to perform system housekeeping and common utility functions. (Indeed, they are sometimes referred to as CUSPs, commonly used system programs.) The system manager can use the programs to monitor and regulate system usage. Some library programs can be tailored by altering the source statements supplied by Digital and recompiling to replace the current copy on the system disk.

File Processor Buffering

The optional file processor (FIP) buffering module accelerates file processing on the RSTS/E system. The module reduces the number of accesses to disk by maintaining more than one disk directory block in memory. The system manager can enhance FIP buffering by allocating additional memory to extended buffer space for use as a cache for disk directory blocks.

▪ RSTS/E Supports Software Disk Caching of File Directories and Data Blocks

Software disk cache is a dynamically allocated portion of main memory in which blocks of disk-accessed file data are stored. When a request is made to read a disk block, the operating system first checks the cache. If the block of data is there, a physical disk access is avoided. This results in faster program execution because disk accesses are minimized.

When the cache is full, new information is read into an area that contains the least recently used block of data. This automatic mechanism ensures that frequently used blocks of data remain in the cache.

Cache size is determined by the system manager. The system manager can designate specific files for caching, or can specify that all files be cached. The system manager can also enable or disable caching of file data, independent of caching disk directory blocks. This system-level capability gives the system manager a powerful method of tuning system performance.

▪ RSTS/E Has System Utility Programs for Both the System Manager and General User

Some system management utilities are privileged programs and can be run only by the system manager or privileged users. Other utilities are not privileged and can be run by the general user, but have privileged features that can be executed only by the system manager.

System management utilities include — system initialization and maintenance programs, resource management and accounting programs, system error logging and analysis programs, operator services and spooling programs, and user communication programs.

Disk File and Device Backup

RSTS/E provides the ability for total or selective backup of accounts and files to disks or to magnetic tapes using DCL commands. This is done through the use of multivolume container files that may be placed on standard RSTS/E format disks or ANSI labeled magnetic tape. A separate utility is provided for making image copies.

Selective backup is done online. Image copies of disk volumes can be made online for all volumes except the system disk and offline for all volumes including the system disk.

RSTS/E BACKUP produces BACKUP sets that are subset-compatible with the VAX/VMS BACKUP and can read BACKUP sets produced by VAX/VMS BACKUP. This provides for easy transfer of data between machines running these two operating systems.

General System Utility Programs

RSTS/E provides several utility programs available to the general user. These programs include system information and terminal utility programs, file utility programs, and special service programs. Like the system management utilities, they are stored in the system library account and are called and executed by issuing the RUN system command or, if it is available, the appropriate CCL command.

The list of programs that follows is not exhaustive; it is meant only to suggest the range of utilities available under RSTS/E operating systems.

Special Service Programs

MAC MACRO	Assemble MACRO-11 source code into object format. MAC operates under the RSX-11 runtime system; MACRO operates under the RT-11 runtime system.
LINK	Link object modules produced by FORTRAN or MACRO into an executable image that runs under the RT-11 runtime system.
TKB (Task Builder)	Builds an executable image by linking object modules produced by the MAC assembler or language processors other than FORTRAN. The resulting task image runs under the RSX runtime system specified by the user.
RUNOFF	Generates a formatted listing of a text file containing special RUNOFF text formating commands.

Security

Security is a concern of anyone using a commercial or administrative system. The RSTS family has evolved a strong and well tested security system that provides system and data protection at every level. Access can be controlled in many ways. Systemwide passwords and hashed passwords of up to 14 characters control the access to accounts. Special accounts called "captive accounts" automatically start up an application when the user logs into the system and logs the user out when the application is exited. A multiprivilege system assures that users can perform only those functions permitted by the system manager. Access to data files and programs is controlled by protection codes that specify who is authorized to use them. And, finally, system resource usage is controlled by quotas.

Privileged Capabilities and System Operation
A program is privileged when it is an executable file and has a protection code of <192> or greater. Only the system manager or other users running under privileged accounts can create or modify privileged programs.

Security and Privilege
Access to the system is controlled by using passwords that can be 6 to 14 characters in length. Initial assignment of passwords is done by the system manager. Each user, given the required privilege, can change his or her account password. Passwords are normally stored in hashed form. This feature is available on a per account basis at the option of the system manager.

The system manager may optionally define a system password that must be entered before the user is prompted to login to an account. This feature is available for different classes of access such as dialup or network access.

As a resource sharing system, RSTS/E can give every user access to the system peripherals and resources, as well as a wide range of additional capabilities. Usage restrictions can be imposed on a per user or per system basis on certain of these resources.

RSTS/E provides 34 separate privilege attributes that can be assigned to any account on the system. Tasks that are resident on the system may require one or more of these attributes in order to run. In the latest RSTS/E system version there is no connection between the numbers that designate accounts (PPNs) and the privileges assigned to the account.

Account Management

DCL commands are provided to create and delete accounts, set account attributes, and display account information. Account templates can be created and used to set the defaults for a class of accounts.

Several different types of accounts can be defined. User accounts allow access to the RSTS/E system. Captive accounts cause the system to startup a specific application such as word processing or a menu. When the user leaves the application, the job is killed and the user is logged off. Guest accounts can be set up causing the system not to prompt for the password during login. Accounts can be designated as noninteractive, causing the system to disallow a user to login.

Unlimited Access

No file in the RSTS/E system can be protected against a privileged job. A privileged job can create and delete files under any account number on any disk. Such unlimited access does not generate the normal PROTECTION VIOLATION error.

▪ Batch Processing Lets You Submit Jobs without Using Terminal Dialog

BATCH is particularly useful in executing large data processing operations for which interactive requirements are not a factor. Batch input can be submitted as standard DCL command files. It is possible to execute multiple streams simultaneously by running multiple copies of the BATCH program. The capabilities of BATCH are controlled by the system manager.

Chapter 5 • RT-11

▪ RT-11 Offers Realtime Versatility and a User-Friendly Environment

Digital provides a compact operating system for realtime, single-user applications—RT-11. RT-11 is well suited to such applications as laboratory and factory instrument control, manufacturing process control, flight management, topography, and numerous other technical jobs. RT-11 systems, in widespread use in commercial applications too, can be found doing word processing, medical record management, computerized estimating for general contractors, and typesetting for newspaper publications.

RT-11 systems offer powerful services in a friendly environment and make efficient use of system resources. For example, the RT-11 operating system supports the standard Digital Command Language (DCL), making access to system services as easy as typing English-like commands. Instead of having to manage system calls directly, you can call services through DCL commands that will prompt for any missing parameters and will offer help if a problem or question arises.

The keypad editor, KED, is especially designed for a wide range of videoterminals, and takes advantage of all their advanced features. Screen-oriented editing lets you see immediately what effect your editing instructions have, and permits quick changes to correct errors or to accommodate altered program needs. The KED is simple to use and easy to learn; even a novice can begin editing right away. The most common editing requirements use no more than two keystrokes and the editing instructions themselves are clearly named to explain what they do.

The RT-11 operating system supports a number of high-level languages including BASIC-PLUS, FORTRAN IV, and FORTRAN-77. RT-11 software runs on a variety of PDP-11 and MicroPDP-11 processors.

Other software features are described later in this chapter to show you the full range of tasks RT-11 accomplishes in making the system accessible to novice and experienced users alike—including CTS-300, Digital's multiuser, commercial timesharing system that combines the DIBOL language with the RT-11 operating system.

- **Requires Minimal System Generation Because Several Preconfigured Packages Are Offered**

Because RT-11 comes in a ready-to-use form, only users requiring special features or a highly optimized system tailored for a particular application have to perform system generation.

- **RT-11 Systems Include a Choice of Monitors**

The monitor is that part of the operating system that controls and allocates the services of the rest of the system. RT-11 systems give you a choice of three different monitors. To accommodate the range of typical RT-11 users, Digital supplies the system with a single-job monitor, a foreground/background monitor, and an extended-memory monitor.

The *single-job monitor*, called SJ, organizes the system for single-user, single-program conditions. Even in configurations with as little as 24 Kbytes of memory, an SJ will not sacrifice access to system programs, services, or features (except extended memory and the use of a foreground or system job). Because the resident portion of the SJ monitor itself needs only 4 Kbytes of memory, it leaves much room for extensive program development. Also, because the monitor services interrupts very quickly, SJ systems are ideal for programs that require a high data transfer rate.

The *foreground/background* monitor, called FB, takes advantage of the fact that much central processor time is often spent waiting for external events such as I/O transfers or realtime interrupts. In FB monitor systems, this waiting time is put to good use by letting you use the central processor for another (background) job while your principal job is pausing. For example, if your foreground monitor is running a laboratory sampling program, your background job could be a FORTRAN program development session, or an accounting file update. When the foreground pauses to wait for data from the instruments, the background lets a programmer work on coding or updating the file.

In FB situations, the foreground job is always of higher priority than the background one, so that the system returns control to the foreground job when it is again ready to run. The FB monitor can set timer routines and send data and messages between the two jobs.

Finally, there is an *extended-memory monitor*, or XM. The XM provides for the simultaneous execution of as many as seven jobs in the foreground and a background job. It allows foreground and background jobs to extend their effective logical program space beyond the 64-Kbyte space imposed by 16-bit addresses on PDP-11 computers. The XM monitor contains all the features of FB plus the capability of accessing as many as 4 Mbytes of memory. Extended memory is managed by the monitor as a system resource, and dynamically allocated to programs as they need it. The extended memory overlay feature permits overlays to reside in extended memory rather than on disk; this results in faster execution time for overlaid programs.

These three monitors are upwardly compatible. FB provides all the services of SJ and XM provides all the services of FB. As system generation options, FB supports system jobs and XM monitors support error logging.

You can access the monitor either through keyboard commands or programmed requests. With keyboard commands, you can load and run programs, start or restart programs at specific addresses, modify the contents of memory, and assign alternative device names, and more. If you have a series of keyboard commands, you can create an *indirect command file*, a list of commands to be executed sequentially by the computer. Typically, you would write an indirect file for jobs that use a lot of computer time but do not need your supervision or intervention. Another possible use of indirect files is for command sequences that you use repeatedly and that are time-consuming to retype. Simply construct the indirect file and call it up when you want the monitor command procedures it includes.

Programmed requests to the monitor appear in the System Macro Library (SYSMAC.SML) and permit an assembly language program to access monitor services. Once a program is running, it communicates to the monitor through programmed requests. If your programs are written in a language that supports the FORTRAN call interface, they can access the monitor services through the use of routines contained in the System Subroutine Library (SYSLIB.OBJ). Typical programmed requests manipulate files, perform input and output, and suspend or resume program operations.

▪ User Interface

RT-11 provides three supported command processors. They include the Digital Command Language (DCL), Concise Command Language (CCL), and User Command Language (UCL). Also provided is an interface for a user-written processor (UCF). The UCF facility permits a user to preprocess commands before they are passed to DCL, CCL, and UCL.

I/O Commands

RT-11's implementation of DCL is a set of keyboard monitor commands that include wildcards, factoring (a method of simplifying string replacement), abbreviations, and prompts. Here is a short example of prompting:

COPY/CONCATENATE

From? DX1: (TESTA, TESTB)

To? DX2: TEST.LST.

The system continues to prompt for input and output file specifications until you provide them. Keyboard monitor commands can be collected together into direct command files.

With a Concise Command Language (CCL) command, you can use run and pass arguments automatically to designated programs stored in the system library. The programs can be system utilities supplied with the operating system, or user-written programs that perform application operations specific to your job.

The CCL commands not only provide an easy-to-use command interface but they can also offer protection from unauthorized users accessing certain programs. For example, if a particular program performs several operations, some of which should not be available to unauthorized users, the system manager can prevent those users from issuing the RUN command to run the program, but can allow them to perform the safe operation subset by using CCL commands.

▪ Programmed Requests Provide Access to System Services

The RT-11 operating system permits access to numerous services through programmed requests. A programmed request inserted directly into the program provides the mechanism.

Under the RT-11 system, programmers can use programmed requests or system library calls to perform file manipulation, data transfer, and such other system services as loading device handlers, setting a mark time for asynchronous routines, suspending a program, and calling the Command String Interpreter (CSI).

▪ Logical Disk Subsetting and the Virtual Memory Disk Handler Provide File Structure and Access Method Capabilities

Using Logical Disk Subsetting (LD), the RT-11 file system with its contiguous files facilitates fast access and ease of use. RT-11 systems can divide user files into logical disk subsets. This facility lets you define subsets of physical disks. Operations can then be performed as though the logical disk were a physical disk. This feature allows more directory entry space and enhances device and file operation.

The Virtual Memory (VM) Disk Handler allows memory above 56 Kbytes to be used as though it were a disk device. That virtual device can be used as the system volume or as a data volume.

▪ Integrity/Reliability Features

The Error Logger (EL) monitors the hardware reliability of the system. It keeps a statistical record of all I/O operations on devices that call it. At intervals that you determine, the error logger produces individual or summary reports on some or all of the errors that have occurred. Support for the error logger must be obtained through system generation. The error logger runs with the FB or XM monitors, either as a foreground or as a system job, but is not supported under the SJ monitor.

▪ System Utilities

System utilities allow you to help system users interact with the system, develop applications and manage system resources. They help you to tune system performance and develop programs faster and easier.

PDP-11 operating systems provide, in general, three kinds of system utility programs — program development utilities, file management utilities, and special system management utilities.

Most *system management utilities* included in an operating system depend on the function the operating system serves. For example, RSX-11M, RT-11, and RSTS/E include system error logging and report programs. RSTS/E and RSX-11M-PLUS include user accounting programs. For additional information, see Chapter 17, "File Management Utilities".

RT-11 Offers a Choice of Text Editors

One of the most useful of all utilities is a text editor. It makes the job of creating, correcting, and updating programs and files a simple matter of using commands at the terminal. RT-11 provides a selection of editors, including EDIT and KED.

The EDIT program creates or modifies ASCII source files and prepares them as input to other programs, such as compilers or assemblers. This program can read files from any input device and write them on any output device. It is a line-oriented editor, most useful for hardcopy terminals.

EDIT performs the four functions that you would expect from an editing program:

- Locates the text to be changed.
- Executes and verifies the changes you command.
- Lists a copy of the edited page on your terminal.
- Outputs the page to the output file.

In order to process text, EDIT thinks of a file as divided into logical units called pages of about 50 or 60 lines in length. Such a logical page corresponds roughly to a physical page in a program listing. One page at a time is read from the input file to the buffers, where it becomes available for editing.

The KED is a videodisplay editor meant for use with VT100 and VT100-compatible terminals. It takes advantage of the special keypad to the right of the terminal's keyboard. Keypad keys provide quick operation of various editing functions, such as moving the cursor left or right by a word or character. The KED has a HELP file that makes keypad editing easy to learn.

The virtual KED (KEX) is available for use under the XM monitor only. KEX editing commands are identical to KED commands. However, KEX maximizes the amount of high memory used while minimizing the amount of low memory used. Therefore, KEX will continue to operate in many instances where there is insufficient low memory for KED to run. In addition, KEX can be run as a foreground or system job, allowing editing to continue while the background is performing some function. On systems that include multiterminal support, multiple copies of KEX may be run (by using the SRUN and FRUN commands), each from its own terminal.

Single-line Editor (SL)
This feature lets you edit the current keyboard command line or CSI command string typed on a videoterminal prior to terminating the line. The previous commands or input lines can also be recalled for editing.

Hardware Characteristics Program (SETUP)
The SETUP program sets terminal (video and hardcopy), lineprinter, and system clock modes using English language commands. SETUP commands are especially useful when you include them in startup indirect command files or IND control files.

Transparent Spooling Package (SPOOL)
The SPOOL is a utility device you can use for sending files to any RT-11 device. Although SPOOL is especially useful for spooling files for printing, the output device is not restricted to the lineprinter, but must be a serial device. SPOOL is distributed with two default output devices, LP and LS, but you can change the default output by customizing your system.

SPOOL is transparent. Once running, SPOOL automatically intercepts all data directed to the lineprinter or other designated output device, stores it, then forwards it to the lineprinter or output device. You can send output to the lineprinter, explicitly by typing a command such as COPY MYFIL.MAC LP:, or implicitly by typing a command whose default is to send output to the lineprinter, such as MACRO/LIST MYFIL. In either case, you need not type a specific command to spool output.

Another major advantage to using SPOOL is that it begins sending output as it is received and therefore output gets sent faster.

Queue Facility
With the file queuing package (QUEUE and QUEMAN), you can send files to any valid RT-11 output device. Although the Queue Package is particularly useful for obtaining hardcopy listings of files, queuing is not restricted to a lineprinter or to other serial devices. The QUEUE program runs with the FB or XM monitors, either as a foreground or as a system job. QUEMAN, the user interface to QUEUE, runs in the background.

Global Regions in Extended Memory

RT-11 systems include support for creating global regions in extended memory. Global regions are areas of extended memory that are controlled by the operating system, rather than by a particular program. The following are some of the features:

- Global regions can remain allocated in extended memory after a program using the global region has exited.
- More than one program can get access to data stored in global regions.
- You can create as many as ten global regions. A program can attach to a combination of as many as six local and global regions.
- Programmed requests control the allocation of global regions in extended memory.
- User programs, system utilities, system device handlers, and monitors can create global regions.
- A permanent global region cannot be eliminated. For example, the input/output page resides in a permanent global region and cannot be eliminated.
- Global regions use the same programmed requests and general format as regions local to programs. However, additional arguments to the programmed requests are required. The window mapping and input/output definition blocks are the same. The general structure of the region definition block is the same; however, it contains two additional bytes. Global regions use the same directives as local regions.

Virtual Terminal Communication Package (VTCOM)

The virtual terminal communication package (VTCOM) utility lets you communicate with a host system while you run the RT operating system, making your stand-alone system a local terminal. With VTCOM, you can use resources available on host systems, such as electronic mail and programming languages, and still use RT-11 resources. You can also transfer ASCII and binary files between the host and your RT-11 stand-alone system.

The virtual terminal communication package is the software that lets you take advantage of these features. However, VTCOM requires certain hardware components.

RT-11 Systems Have File Transfer Program

TRANSF.SAV transfers binary files between an RT-11-based host system, on which TRANSF.SAV is installed, and the stand-alone RT-11 system. Also provided is TRANSFER.EXE for binary file transfers between RT-11 systems and VMS host and TRANSFER.TSK for binary file transfers between RT-11 systems and RSX host. TRANSFER, although provided in the RT-11 distribution kit, must be installed on the host system.

Ethernet Handlers

The Ethernet handlers provide the user with a common interface to Ethernet class controllers for user-written software. The NC Ethernet handler supports the DECNA Ethernet controller for the Professional 300 series processors. The NQ Ethernet handler supports the DEQNA Ethernet controller for Q-bus processors.

The System Subroutine Library Lets You Access the Monitors

The RT-11 System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines that allows a FORTRAN user to access various features of RT-11 foreground/background (FB) and single-job (SJ) monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and two-word integer support. This library file is resident on the system device (SY:) in the default library that the linker uses to resolve undefined globals and is resident on the system device (SY:).

Some of the functions provided by SYSLIB are

- Complete RT-11 I/O facilities, including synchronous, asynchronous, and completion-driven modes of operation. FORTRAN subroutines can be activated upon completion of an input/output operation.
- Timed scheduling of asynchronous subjobs (completion routines). This feature is standard on FB and optional on the SJ monitor.
- Complete facilities for interjob communication between foreground and background jobs (FB and XM only).
- FORTRAN interrupt service routines.
- Complete timer support facilities, including timed suspension and time-of-day information. These timer facilities support either 50- or 60-cycle clocks.
- All auxiliary input/output functions provided by RT-11, including the capabilities of opening, closing, renaming, creating, and deleting files from any device.
- All monitor-level informational functions, such as job partition parameters, device statistics, and input/output channel statistics.

- Access to the RT-11 Command String Interpreter (CSI) for acceptance and parsing of standard RT-11 command strings.
- A character string manipulation package supporting variable-length character strings.
- INTEGER+:4 support routines that allow two-word integer computations.

SYSLIB allows a FORTRAN IV or FORTRAN-77 user to write almost all application programs completely in FORTRAN with no assembly language coding.

Linker

The RT-11 Linker (LINK) converts object modules produced by an RT-11-supported language processor into a format suitable for loading and execution. The linker processes the object modules of the main program and subroutines to

- Relocate each object module and assign absolute addresses.
- Link the modules by correlating global symbols that are defined in one module and referenced in another.
- Create the initial control block for the linked program that the GET, R, RUN, and FRUN commands use.
- Create an overlay structure if specified and include the necessary runtime overlay handler and tables.
- Search libraries specified by the user to locate any unresolved globals.
- Automatically search a default system library to locate any remaining unresolved globals.
- Produce a map showing the layout of the executable module.
- Produce a symbol definition file.

The linker runs in a minimal RT-11 system of 24 Kbytes of memory; it uses any additional memory to facilitate efficient linking and to extend the size of the symbol table. The linker accepts input from any random-access device on the system; there must be at least one random-access device (disk or DECtape) for memory image or relocatable format output.

Librarian

The Librarian Utility Program (LIBR) lets you create, update, modify, list, and maintain library files. A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO assembler can access them rapidly. The modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience.

RT-11 Offers Four Ways to Change or Debug Your Programs

Four RT-11 programs help you debug programs and make changes to programs that are already assembled. They are the Online Debugging Technique (ODT), Save Image Patch Program (SIPP), the Object Module Patching Utility (PAT), and the Source Language Patch Program (SLP).

- RT-11 ONLINE DEBUGGING TECHNIQUE

RT-11 Online Debugging Technique (ODT) is a program supplied with the system that aids in debugging assembly language programs. From the terminal, you can direct the execution of programs with ODT. The ODT performs the following tasks:

- Prints the contents of any location for examination or alteration.
- Runs all or any portion of an object program using the breakpoint feature.
- Searches the object program for specific bit patterns.
- Searches the object program for bytes that reference a specific word.
- Calculates offsets for relative addresses.
- Fills a single word, byte, or block of bytes with a designated value.

- SAVE IMAGE PATCH PROGRAM

The Save Image Patch Program (SIPP) lets you make modifications to any RT-11 file that exists on a random-access storage volume. It is especially useful for maintaining .SAV image files. Using SIPP, you can examine locations within a file. You can use SIPP from the console, an indirect command file, or a BATCH stream to patch all files created with the linker. SIPP can patch overlaid files created with the Version 4 linker, and nonoverlaid files created by previous linkers.

SIPP was designed to replace the former PATCH utility. However, PATCH is included in the Version 4 release so that patches that have been published prior to this release can be installed.

- OBJECT MODULE PATCH UTILITY
  The RT-11 Object Module Patch Utility (PAT) lets you patch or update any code in a relocatable binary object module. The PAT does not permit examination of the octal contents of an object module; that is a function of SIPP. An advantage to using PAT is that relatively large patches can be added to an object module without performing any octal calculation. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module.

- SOURCE LANGUAGE PATCH PROGRAM
  The SLP is a patching tool you can use for modifying source files. When used with SRCCOM, you can patch a source file so that it will match an edited version. SLP accepts as input a source file you wish to patch and a command file created by SRCCOM.

System Utilities Offer File Transfer, Backup, and Protection for Efficient System Management and Maintenance

Though you may call most of the system services through keyboard monitor commands, the keyboard monitor itself does not perform the work. Instead, it passes the requests to the appropriate system utility programs. By providing this simple interface, the monitor reduces the complexity of programming or operating the computer, which increases your productivity. Some of the utility programs and their functions are listed below.

- PERIPHERAL INTERCHANGE PROGRAM
  The Peripheral Interchange Program (PIP) is used for file transfer and file maintenance utilities. PIP is used to transfer files between any of the RT-11 devices and to merge, rename, and delete files. With PIP you can perform numerous operations simply and avoid the difficulty usually associated with file transfer and manipulation requirements.

  PIP allows you to protect files against accidental deletion. File protection is indicated by the letter "P" next to the file size as listed in the file's directory. Files may be protected and unprotected only by using the RENAME keyboard command or the PIP utility.

- DEVICE UTILITY PROGRAM
  The Device Utility Program (DUP) is for device maintenance. The DUP creates files on file-structured RT-11 devices. It can also extend files on certain file-structured devices (disks and DECtape), and it can compress, scan for bad blocks, image copy, initialize, or boot RT-11 file-structured devices.

- DIRECTORY PROGRAM
  The Directory Program (DIR) performs a wide range of directory listing operations. It can list directory information about a specific device, such as the number of files stored on the device, their names, and their creation dates. DIR can list details about certain files, too, including their names, file types, and sizes in blocks. DIR can also print a device directory summary, and it can organize its listings in several ways, such as alphabetically or chronologically.

- COMMAND STRING INTERPRETER
  The Command String Interpreter (CSI) is the part of the RT-11 system that accepts a line of ASCII input (usually from the user at the console terminal) and interprets it as a string of input specifications, output specifications, and options for use by a system utility program.

- DUMP
  DUMP is the RT-11 program that prints on the console or lineprinter, or writes to a file, all or any part of a file in octal words, ASCII characters, or Radix-50 characters. DUMP is particularly useful for examining directories and files that contain binary data.

- FILE EXCHANGE PROGRAM
  The File Exchange Program (FILEX) is a general file transfer program that converts files from one format to another so that they can be used with other operating systems.

- SOURCE COMPARE PROGRAM
  The Source Compare Program (SRCCOM) compares two ASCII files and lists the differences between them. SRCCOM can either print the results or store them in a file. SRCCOM is particularly useful when it is necessary to compare two similar versions of a source program. A file comparison listing highlights the changes made to a program during an editing session. SRCCOM can also produce (as output) a file of SLP commands that can be used later with SLP to patch an original file to match its edited version.

- BINARY COMPARE PROGRAM
  The Binary Compare Program (BINCOM) compares two binary files and lists the differences between them. You can direct BINCOM to print the results of the comparison at the terminal or lineprinter, or store them in a file. It can also create an indirect command file that invokes SIPP to patch one save file to match another version.

- RESOURCE PROGRAM
  The Resource Program (RESORC) examines the currently running RT-11, system and displays information about the status of the monitor and the system configuration.

- INDIRECT CONTROL FILE PROCESSOR LETS RT-11 RUN UNATTENDED
  The indirect control file processor executes indirect control files. IND control files contain IND directives, that control the execution of the indirect control file. IND control files can also contain keyboard commands (DCL, CCL, and UCL). You can use indirect control files to access other files, execute keyboard monitor commands, define symbols, pass parameters, and perform logical tests.

  When running under the XM monitor, IND stores context information in a region of high memory, resulting in a performance improvement.

- VIRTUAL MEMORY (VM) HANDLER
  The RT-11 system supports memory above 56 Kbytes (and as many as 4 Mbytes on Q-bus systems) as if it were an RT-11 file-structured, random-access device. The Virtual Memory (VM) Handler can be used as the systems device or a data device under the SJ and FB monitors. However, under the XM monitor it can be a data device only.

- SYSTEM JOBS
  In addition to the normal foreground and background jobs, both the FB and XM monitors can support as many as six extra jobs, called system jobs. Digital provides five utilities that can be run as system jobs — SPOOL, VTCOM, KEX, QUEUE, and ERROR LOGGER, EL. System job support is included in the distributed XM monitor and is also available under the FB monitor through system generation.

- ".FETCHable" HANDLERS UNDER XM
  A SYSGEN option allows handlers to be loaded and released dynamically by user programs under the XM monitor. Previously, all handlers needed by user jobs had to be permanently resident in memory, even if they were not being used. Background user programs can now issue the .FETCH and .RELEAS requests to load handlers into memory for use and release when finished, thereby freeing valuable low memory space for other uses.

- WRITE PROTECT FOR DISKETTES
  You can write-protect a diskette by using the "diskette-write-protect" feature via a SET command to the handlers.

- BACKUP UTILITY PROGRAM
  With a backup utility program, a user can quickly store a large volume or file on a set of several smaller volumes, even if the file is larger than one of the smaller volumes. The Backup Utility Program (BUP) lets you initialize backup volumes, obtain directory information about a set of backup volumes, and RESTORE a volume or file from a set of backup volumes to its original form either as a file on a volume or as an entire volume. The BUP does not produce RT-11 structured files when backing up and therefore the information on the backup volumes cannot be accessed as such by other RT-11 utilities. The RESTORE facility of BUP recreates the RT-11 file structure. This very fast, multivolume backup/restore facility supports the streaming capabilities of Digital's nine-track 1,600-bit-per-inch tape drives, the TSV05 and TU80.

Program Development

The MACRO language is included with the RT-11 operating system as the primary means to develop programs. For a complete description, refer to Chapter 11, "MACRO."

▪ RTEM-11 Lets Programs Developed on RSX-11 and VAX/VMS Run on RT-11

The RTEM-11 emulator is an option on Micro/RSX, RSX-11M-PLUS, RSX-11M, and VAX/VMS that provides an RT-11 program-development environment on those systems. Several users can develop RT-11 applications concurrently on an RSX-11 or VAX/VMS host system. Users can create, edit, assemble, and link programs using RTEM-11 and then execute these programs on an appropriately configured RT-11 system. Programs developed on RTEM-11 execute the same way as if they had been developed on RT-11.

Many programs developed on RTEM-11 can also be debugged and tested on that system, although the execution environment supplied is foreground/background only. Examples of software that can be built, but not debugged or executed on RTEM-11, are programs that access the I/O page and any program that requires the XM monitor.

Utilities for RTEM-11 include

- A *File Interchange Program* (FIP), that lets the RTEM-11 user transfer between RT-11 volumes and host volumes.
- A *Jack of All Trades* (JOAT), that performs system and device operations. For example, while most program development can be done using just the system device, console terminal, and lineprinter, other peripheral devices may be required. A JOAT permits access to any host device supported by RTEM-11. A JOAT can also be used to terminate RTEM-11, pass command lines to the host, and show current device usage.
- A *User Command First* (UCF) can implement Digital's Command Language (DCL) for FIP and JOAT.

▪ CTS-300 Offers Business Applications on RT-11

CTS-300 is a complete software environment layered on top of RT-11. It uses DIBOL, a popular, friendly programming language especially designed by Digital for writing business applications.

CTS-300 provides an operating system, a higher-level programming language, system utilities, a text editor, a sort program, and program development tools. Program development may be done in a timesharing environment. CTS-300 is capable of supporting as many as 12 users or up to 16 jobs simultaneously, depending on your application programs.

Depending on hardware and system generation options, suitably configured computers can accommodate a mix of application and program development terminals. In addition, if the hardware includes VT52s, VT100s, or VT200-series videoterminals, the DECFORM screen handler running as part of CTS-300 will take full advantage of the VT100's wide range of features. VT200-series terminals must be in either VT52 or VT100 mode.

CTS-300 gives you immediate access to online information. Interaction with the system comes through high-speed videoterminals where all of the resources of the system are at your disposal for running programs. User programs may perform any data-processing job from inventory control to accounts receivable.

Programs are not swapped in and out of memory while running, but reside in memory in dynamic partitions. This creates two major benefits — fast terminal response time and a smaller resident operating system.

▪ Concurrent Program Development

A major capability of CTS-300 is made available through the CRT-oriented programming editor, DKED. With this feature, it is possible to create and modify DIBOL programs online, with multiple copies running under Extended Memory Time-Shared DIBOL (XMTSD). Further, one or more of the developers could be remote, using dialup lines under XMTSD.

Concurrent program development and application execution provide excellent flexibility. XMTSD runs in either the background or in the foreground. When it runs in the foreground, the background partition is available for program development.

There are two ways of using the background. The first is with one application programmer occupying the background for program development using familiar utilities. Simultaneously, XMTSD can execute applications in the foreground. The second mix occurs when more than one programmer is doing development work in the foreground. This is accomplished by running the editor DKED in the foreground partition as a job under XMTSD. When a program is ready for compiling and linking, it is then submitted to a command file in the background queue. When as many as 16 requests from various programmers are held, the requests are executed in the order submitted.

▪ Ease of Programming

The FLAGS subroutine permits a DIBOL application to select as many as eight independent options at runtime with one optional argument. During the running of an application, it is often desirable to change one of the eight options. This is accomplished by using an optional second argument to FLAGS.

In routine operator/CRT interaction, it is often desirable to time such things as messages displayed on terminals. For instance, information displayed as part of a program could take some time for an operator to read before the next application subset could occur. The SLEEP statement under TSD/XMTSD monitors will defer execution of a job for a specified period of time, thus allowing programmers to schedule time better among the other jobs.

The *Print Utility* gives you a tool to produce simple, ad hoc reports from data files. This utility lets you search a file index for report creation while leaving the master file intact. It is unnecessary to sort the master file first, in a different order, then create a report; a small sorted index file will be sufficient. The enhanced Print Utility can improve programmer productivity.

The *ISAM Utility* allows end-user operators to create ISAM files without operator intervention. The autocreate feature can be initiated directly at a terminal or indirectly via chaining. Faster and easier file reorganization is the result.

The *Linker* in CTS-300 is an easy-to-use utility that converts the output from the DIBOL compiler into a runtime format. The Linker allows a main program to be combined with many compiled external subroutines into a single executable job, and lets you specify overlays. (Overlays allow the runtime job to use less main memory than would otherwise be required.)

The *Peripheral Interchange Program* (PIP) in CTS-300 allows either ASCII or binary files to be transferred among any RT-11 supported devices. It performs directory operations, renames files, extends the size of a given file, and consolidates empty files into one contiguous space. Segmented files can be merged during a PIP transfer, or multiple transfers may be executed in response to a single keyboard command.

The *File Exchange* (FILEX) program in CTS-300 provides a universal file format via floppy disks between Datasystem 300s. FILEX creates an IBM-compatible floppy in 3741 format that can be read by either IBM systems or other DEC Datasystems.

The *Device Utility Program* (DUP) in CTS-300 performs general utility functions in support of disk devices. Among DUP functions are initializing devices, scanning for bad blocks, and compressing data on a disk.

The *Directory Program* (DIR) in CTS-300 is used to list the file directories for disk devices. DIR allows directory listings to be sorted by file name, file type, date, size, or position.

The *Librarian* in CTS-300 creates and maintains libraries of commonly used programs. Each library has a catalog, listing the sections with sufficient information to enable the Linker to find them. The Librarian's tasks include creating libraries; adding new modules to existing libraries; copying options, including selective deletion and replacement during the copy; and listing the catalogs of libraries.

▪ Command Language

CTS-300 is designed to be use interactively. The keyboard commands are consistent in format and easy to understand, and the high-level command language allows transition from source code to executing code with only one statement. CTS-300 also features indirect command files that permit you to invoke a whole stream of system commands, similar to a BATCH stream, by issuing a single, one-word command. There is no complicated job control language to learn, like BATCH stream. Indirect command files accept the same monitor commands that users type at the terminal.

Data Management Services

Data Management Services for CTS-300 (DMS-300) can handle sequential, random, or ISAM-structured files. DMS-300 also supports file sharing and multivolume files. Multivolume file support permits one file, extending over several disk drives, to be processed sequentially or by keyed access, without requiring special programming.

Optional Software

Optional communications, business applications, and report writing packages are available for CTS-300—including DECtype, DIBS-11, and QUILL.

Remote Data Communications Package (RDCP)

The RDCP is a powerful batch data communications package that uses both IBM 2780 and 3780 protocols. It is easy to use, yet gives the data processing center the control necessary to manage a network of systems from a single location. It can be used to communicate with IBM processors as well as other Digital Datasystems. Its key capabilities are as follows:

- Concurrent operation with other application programs, provided the system is configured with 28 Kbytes of memory.
- Unattended operation, depending on hardware configuration.
- Batch command stream.
- Recovery restart.
- Data compression/decompression.
- Transparent data.
- Automatic calling, depending on hardware configuration.
- Autodial or autoanswer depending on hardware.

▪ DECtype-300

DECtype-300, word processing software for CTS-300, provides concurrent word and data processing. It is intended for use by organizations that have a primary need for data processing. Systems equipped with DECtype-300 give users at every terminal word or data processing as needed.

DECtype-300 is a word processing application that produces documents. As you type text from a keyboard, DECtype-300 displays it on a videoscreen. Once the text is typed, DECtype files (saves) the document on a storage device, making it available for later use. At any time you can call the text back to the screen and use the keyboard to modify the document. You can tell DECtype-300 to print the document on any of several printers that may be attached.

The many DECtype-300 options let you produce documents that present your ideas as you intended them. You can highlight sections of text by underlining or bolding them. You can type two or more characters in the same place on the paper, producing composite characters. You can use subscripts and superscripts for footnotes or mathematical formulas. You can right- or left-justify text, or center text between the margins. You can insert tabs between words and margins in any part of the document. You can indent paragraphs, switch between single-spacing and double-spacing, include tables in the text, and produce many other effects.

DECtype's "advanced features" make use of its general purpose computer to meet applications other than word processing. You can insert information from a list document into the blanks in a form letter, producing "customized" letters, run it at the same time as other business applications, and perform simple mathematical calculations.

DECtype is easy for the new operator to use and understand. Wherever you must make major choices, DECtype-300 displays a menu, describing all the options and indicating which letters to type to select each option. Before deleting information, DECtype either asks for confirmation or gives you a second chance to recover the information. When you give DECtype-300 an invalid command, it displays an understandable message telling you how to correct the problem.

DECtype is also optimized for the experienced user. The actual keystrokes required to invoke an option are minimal and easily memorized. DECtype-300 automatically wraps text so that lines of text will fit within the indicated margins, freeing you to look at the source document instead of at the screen. If you should change the margins later, DECtype-300 automatically refits the text to conform to the new margins. DECtype ensures that questions of formatting do not slow down an experienced user during text entry.

Chapter 6 ▪ Digital Standard MUMPS

▪ A Productive Way to Develop Applications

Digital Standard MUMPS (DSM-11) is a multiuser data management system that consists of an interactive, high-level interpretive language (MUMPS), a data management facility, and a timesharing executive.

DSM is well suited for interactive applications that require a large shared database. This need is present in health care, where DSM has been used to develop applications such as patient registration and billing, medical record management, and appointment scheduling. While DSM was developed in a medical environment, use of DSM in banking, transaction processing, electronic mail, inventory control, trust management, and other areas of commerce and industry is gaining in popularity.

Advanced capabilities have been built into the language to provide a high-productivity development environment. These capabilities are in the areas of reliability, performance, productivity, capacity, distributed access, and family compatibility.

The MUMPS language is an extension of the ANSI Standard Specification (X11.1-1984) for Massachusetts General Hospital Utility Multiprogramming System (MUMPS). MUMPS was originally developed at the Laboratory of Computer Science at Massachusetts General Hospital and was supported by Grant HS00240 from the National Center for Health Services Research and Development. System capabilities are heavily oriented toward string manipulation and relieve the user of any concern for programming peripheral devices or for structuring databases in the traditional sense.

Language processing by the system is interpretive. This greatly facilitates program development by eliminating the need to load editors, assemblers, or linkers. The DSM-11 application programmer need only be concerned with developing the proper logical hierarchy for a database and efficient logic for the application requirements.

In addition to supporting the Standard MUMPS language and providing all operating system capabilities, DSM-11 affords a unique database structure. Data, referred to symbolically, is automatically stored and linked in sparse, hierarchical structures called M-trees. The physical and logical allocation of mass storage for the tree-structured database is handled completely by the operating system so the programmer can concentrate on application data relationships. A database can be made available to all system users or be restricted to a class of users.

DSM-11 is especially suited for environments requiring high productivity, superior performance, and distributed data processing. DSM-11 operates on the full-range of PDP-11 family of computers, from the MicroPDP-11 to the PDP-11/84 and is completely compatible within the PDP-11 family making system upgrades easily achievable.

DSM-11 offers the following features:

- Conforms to ANSI MUMPS specifications, with significant additional enhancements.
- Provides a powerful symbolic debugger that helps modify programs quickly and efficiently.
- Enables programmers to write error handling routines at every execution level.
- Enables frequently used routines to be stored in memory.
- Supports as many as 4 Mbytes of memory (2 of which can be used for cache buffers).
- Enables modular expansion through distributed data processing (DDP) and local area networking (LAN) capabilities.
- Offers powerfail/restart capabilities for systems with battery backup.
- Supports magnetic tape streaming.
- Supplies unattended backup to free system resources during peak hours.
- Provides separate logical databases that can be independently mounted.
- Uses an in-memory disk cache to allow efficient sharing of the data among all users.
- Provides dynamic bad-block management for UDA type disks.
- Journals database modifications to either disk or magnetic tape.
- Provides system maintenance utilities such as hardware device error reporting, system patching capabilities, and an executive debugging tool.
- Spools output to devices such as printers.
- Provides streamlined system installation and system generation procedures, identifying all hardware devices automatically during system generation.

System Generation

System generation (SYSGEN) can be easily achieved by the user through DSM-11's autoconfiguration feature. The autoconfigure feature automatically detects the hardware vectors and the control status registers (CSR) for every hardware device that is configured on your system. This data can then be incorporated into your own system configuration file. It's a feature that simplifies the installation process because you don't need to know which hardware devices are connected to your system.

In addition the SYSGEN utility provides an interactive means to create a DSM-11 configuration so that you can customize different hardware configurations. These configurations can be stored on a disk, and the configuration desired can be selected at system startup time.

Job Scheduling

The executive implements the timesharing aspects of the system and permits partitioned multiprogramming using dynamic assignment of memory-resident user partitions. In a timesharing environment, jobs are generally highly interactive and normally require little processing time between I/O requests. The executive passes control from one user to another in order to use the central processor as much as possible. Because jobs are resident in memory partitions, the executive can switch from user to user in minimum time.

I/O Monitor

When a job becomes I/O bound, the executive places the job in the appropriate hung state that signals the I/O monitor to start its processing. The I/O monitor initiates and processes the I/O activity through its interrupt handlers.

The DSM interpreter and the I/O monitor communicate through buffers for terminal I/O character processing, but the I/O monitor supervises the asynchronous filling and emptying of these buffers to overlap output with that program's processing whenever possible.

The I/O monitor creates a terminal-independent environment in which an application program can run with any terminal of the hardware system regardless of its specific speed and formatting characteristics. At terminal log in, a partition initially "owns" one terminal. It may subsequently acquire other terminals in the system or it may release the original terminal and continue as a detached job.

The I/O monitor also supervises the peripheral I/O devices of the system, including magnetic tape drives and lineprinters.

User Interface

DSM-11 users gain access to the system's programs by using a special log-in sequence that involves one or two access codes. These codes, provided by the system manager, are the User Class Identifier code (UCI) and the Programmer Access Code (PAC). Using the UCI allows access to the programs and global variables listed in the program and global directories.

Users who are permitted simply to run programs log into the system by typing in the UCI followed by the name of the program they wish to run.

Users who are allowed to create or modify programs log into the system by typing the UCI followed by the PAC. A programmer can issue DSM commands at the keyboard, as well as create, modify, and delete global data and programs associated with the user's UCI.

If the user intends to write programs, the partition is initialized and control passed to the interpreter. If the user desires to activate a service program, the requested program is loaded from the disk into the partition for execution. In either instance, the user retains the partition until logging-off the system or until the requested program finishes executing.

Data Management

In DSM-11, all data is referenced symbolically, in the context of hierarchical global variables and arrays. The content and structure of the tree-structured global arrays are logically mapped into the system's physical storage medium. The database supervisor maps logical information from global arrays into directories of fixed-size blocks. Maps of unused disk blocks are maintained to facilitate the dynamic allocation of disk storage space to files. These storage allocation maps are bit maps in which there is a correspondence between the map address and the bit position within the map and the disk address of the block.

Whenever a file needs a block, the system references a table that governs the storage allocation of data for that particular user. The table has entries that indicate the block number to start scanning for an empty block.

DSM-11 uses a data retrieval method known as disk cache. Once a block of data accommodating a given level of subscripting is referenced, its address is placed in the partition's overhead area. The block remains in memory until a different block is referenced. Often, no further disk access need be made to reference associated information at the same level. Disk data blocks will be kept in the buffer pool as a function of frequency of use. Frequently used blocks will tend to remain in memory, reducing the number of disk accesses. Also, when data is updated, the individual data elements within a chain are repacked and reorganized to ensure maximum use of space for variable-length data.

A deleted part of a global structure is attached to a garbage chain. The garbage collector program removes the blocks from the tree-structured chain and updates the storage allocation maps accordingly.

Data Storage Elements

All user data, numeric or string, is stored in the system as ASCII character strings. DSM-11 interprets these strings in one of two ways—as numbers, such as those used in calculations, or as strings, such as names and addresses.

Numbers in DSM are signed numbers that can be up to 32 significant decimal digits long. Examples of numbers are

2.08
151.95
403.333
-0.6379465

DSM string data is any contiguous series of legal DSM characters that is to be considered a single data entity. Strings in DSM can be up to 255 characters long. Examples of strings are

HELLO, MY NAME IS
55 SECONDS
2,564,843,485,076,193
FRIENDS, ROMANS, COUNTRYMEN ...
FROP%X10.CF

Variables

Program data values can be expressed as literals, constants, or variables. Two types of variables can be created in DSM programs—local variables and global variables, each of which may be subscripted. Variables can be created, modified, and deleted using the SET, READ, and KILL commands.

A subscript is a value enclosed in parentheses and appended to a variable name. It uniquely identifies data elements that are to reside under that variable name. All subscripted variables residing under a common name are collectively referred to as an array. An array can consist of variables with more than one level of subscripting; when more than one level is used for global array subscripts, they are separated by commas.

DSM uses sparse arrays that contain only those elements explicitly defined. Unlike other languages that may require a declaration of the maximum size of an array to preallocate space, DSM dynamically allocates storage for all array elements only as needed.

Local variables, which are variables that reside in the same partition as the commands that created them, are used as scratch or transient data. They are accessible only to programs running in the same partition. Variables such as ABC, R45, X, %D have no subscripts and are called simple variables. Subscripted variables can have multiple levels of subscripting, with numeric or string subscripts, such as ABC(2), R49("LIST"), or ABC(4 B(C*D)/X,89).

Global variables are subscripted arrays stored on disk. External to a program's partition, they provide a common database available to all programs authorized through the system protection scheme. There is no logical limit to the number of subscripts that can be used. Like subscripted local variables, global arrays also reside in sparse arrays and are created simply by reference in a program. Each global array name is similar to a local variable name, but is always preceded by the circumflex symbol (^).

The DSM-11 Disk Structure and Database Arrays

Disk volumes allocated for the storage of DSM databases and programs are the primary storage media used by the DSM-11 system. Each UCI defined by the system manager has two directories associated with it: the global directory (that is, the file directory) and the program directory.

The system manager can locate the directories on any disk unit in the system and can also limit program and global storage to specific disk units.

Globals are logically organized as multidimensional tree-structured arrays. An element of an array has a logical name consisting of the global name and the subscript or subscripts uniquely identifying the element. For example, ABC(2,3.4,"JONES") is the name of the element in the global called ABC whose first subscript is 2, whose second subscript is 3.4, and whose third subscript is "JONES." The elements of a global array are called nodes. The user's global directory contains the names of all the globals it can reference, as well as pointers to the tree structure for each of the globals.

Sometimes the need arises for large databases that span multiple disks. When this happens, DSM-11 has a simple solution with volume-set capability. Separate logical databases (mountable-volume sets) can support up to eight physical disks on a single CPU and each CPU can support up to four volume sets. Each volume set can be accessed locally or remotely using DSM-11's networking capabilities.

The UCI (User Class Identifier) translation facility lets you create a logical translation table for any global data access in your application system. The translation provides an application-transparent way to access data in another UCI or on another system. The UCI translation facility makes your application much more portable because you're not tied to one particular configuration. Using volume sets with UCI translation offers you the high availability that your application demands by expanding your distributed data processing capabilities.

▪ Terminals and Ancillary I/O Devices

In addition to the disk space reserved by the DSM-11 database supervisor, DSM-11 allows access to terminals and ancillary I/O devices such as the lineprinter and magnetic tape. Each I/O device has a unique identification number in the system.

The OPEN command establishes ownership of terminals and ancillary I/O devices. Then I/O may proceed, using available I/O commands. In general, the programmer need not be concerned with specific characteristics of I/O devices because data transfers consist of ASCII strings not greater than 255 characters. There are, however, certain physical operating characteristics that may be of interest to the programmer, such as rewinding a magnetic tape or form feeding on the lineprinter.

The commands affecting input/output operations to the terminals and ancillary devices are READ, WRITE, WRITE*, ZLOAD, and ZPRINT. The WRITE* command is used to output both local and global data, as well as literals, constants, and format control characters. The WRITE* command is used primarily to take advantage of I/O device special features, which are specified, generally, by nonprinting ASCII codes. The WRITE* command accepts numeric arguments, of which the low-order seven bits are taken as the decimal representation of the ASCII code. For example, the command WRITE *10 is used to output a line feed character.

DSM-11 also has three special "devices." They are the Sequential Disk Processor, the CPU-to-CPU device, and the Job Communication device. The Sequential Disk Processor (SDP) allows the user to physically access the disk as an assignable sequential I/O device. The SDP can access only the disk space that is explicitly set aside for its use. Other disk space, including the global database structure, cannot be accessed. Sequential disk processing allows the user to impose any file structure on the SDP space.

The CPU-to-CPU device is a DMR11 synchronous interface, full- or half-duplex, that connects one DSM-11 CPU to another CPU. The other CPU does not necessarily have to be a DSM-11 system, but does have to recognize DDCMP protocol. This device allows a MUMPS program to communicate with a program running on another central processor.

In-memory job communication permits jobs to send information to other jobs without using the disk. Communication occurs through a series of pseudodevices that are used in pairs — even-numbered devices are "receivers" and odd-numbered devices are "transmitters."

▪ Security Features

DSM-11 provides two security features to help you protect access to data and access to the system. With these features you can control levels of protection for data files, and you can limit the tasks and resources available at certain terminals called "tied terminals."

DSM-11 File Protection Scheme

DSM-11 provides controlled access to the globals in the global directory of each UCI. This allows you to specify privileges for the following user categories:

SYSTEM	Users logged into the manager's UCI.
USER	Users logged into the same UCI in which the global resides.
GROUP	Users logged into any UCI on the volume set on which the global resides.
WORLD	Users logged into any UCI on any volume set, and on any other CPU in the network.

Each user category can be permitted or denied the following types of access:

READ	The right to examine a global.
WRITE	The right to modify a global.
DELETE	The right to kill a global.

Utilities are provided to override the default protection for globals.

Tied Terminals

DSM-11 also employs a concept known as "tied terminals." An attempt to log-in at a tied terminal activates the task to which the terminal is tied and limits the user to the resources associated with that task. This capability gives the system manager an effective control mechanism for system access.

▪ Integrity and Reliability Features

DSM-11 provides several features that help to ensure system reliability and data integrity. These include dynamic bad block replacement for disks, program error logging, journaling of data, database error checking and fixing, and powerfail/restart.

Dynamic Bad Block Replacement

DSM-11 provides dynamic bad block replacement for disks connected via the UDA50 controller. When a bad block is discovered by the operating system, that block is entered into a bad block table on the disk and replaced with a usable block. An error is printed on the system console and the system caretaker records it in a log file.

For non-UDA disks, an error is still reported by the system caretaker; however, the system manager must manually enter a bad block into the DSM-11 Bad Block Table.

Error Logging

DSM-11 provides a software error detection, reporting, and analysis system that logs MUMPS program errors uncovered while running a DSM-11 routine. Each time an error occurs in a DSM-11 application, the location of the error and the local symbol table are stored in a global variable. This allows the programmer to reload the program along with the saved symbol table at a later time for the purpose of debugging.

Journaling

DSM-11 also supports the technique known as journaling, which allows an additional copy of database modifications on the disk to be made on another device. In the DSM system, any item that is changed on the database may also be written to a disk or magnetic tape as a journal record. If a disk failure occurs, it is possible to restore the journal entries onto the previous backup copy and bring the system up-to-date as of the time of the failure. Journaling runs at the system level, and is transparently built into the operating system so that DSM-11 programs do not need to be specially written to handle journaling. The desired database transactions are recorded automatically onto the journaling media.

Integrity Checker

DSM-11 provides a set of utilities that detect and report on logical errors found in a DSM-11 global database.

Database Fixer

DSM-11 provides a tool to reconstruct corruptions found in the logical structure of the DSM-11 global database.

Powerfail/Restart Capabilities

DSM-11 allows powerfail/restart for processors with battery backup. If a power failure occurs, DSM-11 will restart automatically when the power is restored, with your data intact.

▪ Performance Features

DSM-11 provides a high level of performance by means of features such as memory-resident routines, an in-memory "cache" of disk data, and magnetic tape streaming.

You can allocate areas in memory to store frequently used routines within an application. As many as seven separate sets of routines (each of which can hold up to 1,024 routines) can be stored this way and can be executed without the need for direct disk I/O. By using memory-resident routines, dramatic improvements in system throughput can be achieved.

Two Mbytes of memory can be allocated to DSM-11's disk buffer pool. This "cache" can make your application much more efficient because it saves disk accesses for retrieving information that is frequently used, while allowing data to be shared among all the users on your system.

DSM-11 supports magnetic tape streaming, a technique that eliminates mechanical start/stop operations between block reads and writes. It does this by using multiple buffering to write data to the tape at a high, constant rate.

Utilities

A set of utility programs provides the user with the tools to maintain and service the system efficiently. All these utilities are written in the Standard MUMPS language, and as such can be easily modified and extended to suit the needs of a particular installation.

The utility programs consist of two operationally distinct groups:

- System Utilities. The system utility programs provide functions for use by the system manager. They are under the control of the system UCI and are accessible only to those individuals possessing the system UCI code.
- Library Utilities. Library utility programs provide general services that are available to all system users, regardless of UCI. The library utilities include an interactive package that provides online help information and access to the utilities through a series of menus. Library routines perform a variety of services including editors, global management, routine manipulation and reporting, and tasks such as printing date, time, formatted header, cursor control, and conversions.

Unattended Backup

DSM-11's unattended backup process allows disk copying to be performed without user intervention and during off-peak hours. The unattended backup process protects your database while leaving system resources completely available during work hours. In addition, the unattended backup utility provides high-speed disk-to-tape backup when copying complete disk volumes.

Database Restore

These utilities allow you to restore a database that was saved previously with the backup utilities.

Disk Preparation

The disk preparation facility, an interactive set of utilities, provides a means to format, test for bad blocks, and initialize a new DSM-11 disk volume. The user may extend an existing volume set by adding the new disk; or the user may create an entirely new volume set that is logically independent of the running DSM-11 database. Alternatively, the disk volume may be designated for use as a SDP (sequential disk processor) or journal volume.

Volume Set Mount

This utility allows the system manager to mount a volume set on a running DSM-11 system. The volume set is a logically independent disk or disk set that may contain UCIs and data that are accessible from the running system. The mount utility also provides a means to mount journal or SDP disk volumes.

Disk Block Tally

This utility produces a complete report of disk utilization by UCI for a specified volume set.

System Startup

DSM-11 provides all the utilities to reconfigure a booted single-user baseline system into a multiuser customized DSM-11 system. Startup allows the system manager to select any one of the system configuration files created by the system generation (SYSGEN) routine. The system manager can also create a customized Startup File to specify additional activities to take place at startup, including

- Caretaker Startup
- Journal Startup
- Spooler Startup
- Mount Additional Data Volume Sets
- Startup Distributed Data Processing
- Apply System Patches
- Select Error Printer
- Load Mapped Routine Sets

System Shutdown
This utility provides an orderly shutdown of an active DSM-11 system. It brings the system to the point where the processor can be halted to perform various hardware and software maintenance tasks or to bring up a different DSM-11 configuration.

Background Job Detach
Through the Job Detach utility a system manager can spawn a job currently running from a terminal. The job retains its complete symbol table and operating context and frees the terminal. The detached job may be reattached via the Attach Utility.

Background Job Attacher
This system utility allows the system manager to attach to a running job that has been started using the JOB command. The DSM-11 interactive debugger may then be used to insert break points, watch points, or inspect/change the local symbol table.

DSM-11 Spooling
The spooling device is a file-structured mechanism used for temporary storage of information. Typically, a user directs the output of several programs to separate files on this device. The files are then processed one at a time by a despooling program that writes them to an output device.

Each file has a destination code and its own unique file index number that aids in classifying spool files. The destination code is a value, in the range of 1 to 255, recorded in the directory entry for easy access. By using this code, a file can easily aid in retrieving a particular group of files.

These utilities allow the system manager to create or delete disk spool space, allocate a default spool device, and startup or stop a system despooler.

The Symbolic MUMPS Debugger
The DSM-11 debugger retains the current program context, allows single-step program execution, and provides nine adjustable breakpoints. The debugger can be invoked during program execution by typing <CTRL>B. The debugger is entered after the next MUMPS command is executed and the prompt DB> appears. The ZGO command restarts program execution, and a <RETURN> forces single-step execution of MUMPS commands.

DSM-11 Memory Resident Routines

DSM-11 permits the user to set up an area in physical memory to store more frequently used MUMPS routines. A utility provides a means to build a mapped routine set. Once the set is built, it may be loaded into memory. When a MUMPS routine is called, the routine map is first searched before accessing the disk. Routines are executed from the map without copying them into the job's partition, resulting in both disk and CPU performance requirements.

Editors

Among the library utilities DSM-11 offers are two editors for DSM-11 programs. The DSM-11 Editor has a very terse syntax, which makes it ideal for low baud rate terminals. In addition, DSM-11 provides an EDI editor similar to the EDI editor found in other PDP-11 operating systems. This editor is based on a series of commands such as ADD, BEGIN, CHANGE, FIND, and SAVE. It also provides online help text. DSM-11 also provides an editor for global variables. It allows node-by-node edits on one or more globals or parts of globals. The editor allows you to alter only the data associated with a global node, not the node reference itself.

Global Utilities

DSM-11 provides the following global management utilities—Global Management, Global Copy, Global Directory, Global Efficiency, Global List, Global Restore, Global Save, Global Selector, and Global Subscript Filter.

Routine Utilities

The DSM-11 utilities that help you manage routines include Routine Compare, Routine Copy, Routine Directory, Routine Restore, Routine Save, Routine Search, and Routine Cross-Reference.

Miscellaneous Utilities

Other utilities available with DSM-11 include Decimal/Octal Conversion, Show Current Date, Show Current Time, I/O Device Selector, Header Formatter, Menu Manager, and Modem Autodialer.

DSM-11 Communications Capabilities

DSM-11 offers extensive capabilities that let you link computers and terminals into flexibly configured networks. These networks increase the efficiency and cost-effectiveness of your DSM-11 applications. Distributed data processing (DDP) allows multiple DSM-11 systems to share MUMPS database elements transparently. This facility allows a single MUMPS application system to be served by more than one machine. DSM-11 DDP can operate over both DMR11 point-to-point communication links, or Ethernet (DEUNA, DEQNA) multipoint links. DEUNA/DEQNA DDP offers high-performance, low-cost local area networking capabilities, while DMR11 DDP offers both local and remote networking capabilities.

Message Mode Communication
DSM-11 supports error-free message mode communications between processes on different Digital machines using DMR11 synchronous interfaces.

Binary Synchronous Communication
Included with the DSM-11 software is a complete package of system software and utilities that allow the transfer of data between a DSM-11 system and IBM systems. The binary synchronous communication driver can be used to create a protocol emulator for 2780/3780 transmission terminals and 3270 information display systems.

Terminal Communications
DSM-11 supports local or remote asynchronous links between CPUs via terminal lines.

The DSM-11 Language

In addition to being an operating system and data management system, DSM-11 is a powerful, high-level MUMPS programming language with key applications development capabilities. The language is directed primarily toward the processing of variable-length string data, making interactive database systems easier to implement and maintain.

DSM-11 encompasses all the additions made for the 1984 ANSI MUMPS revision. In addition it supports many of the currently specified Type A extensions. It extends the MUMPS language with block structuring so that you can create code that is more readable and easier to maintain.

The DSM-11 Interpreter

DSM-11 is implemented as an interpreter. This minimizes the programmer's time in solving a problem, the computer time needed to check it, and the elapsed time required to obtain a final running program. The interpreter is that part of the operating system responsible for these services.

The interpreter examines and analyzes all Standard MUMPS language statements and executes the desired operations. Each Standard MUMPS language statement undergoes identical processing every time it is executed by the interpreter. Intermediate code is not generated. Comprehensive error checking is also performed to ensure proper language syntax.

In addition, the interpreter stores and loads programs through the disk storage system. During program execution, the interpreter can overlay external program segments invoked by an active program. Proper linkages are set up to return to the invoking program when execution of the segments terminates.

A number of major advantages are obtained from the interpreter. Programs written in an interpretive language do not require any compilation or assembly. Error comments during execution are printed at the programmer's terminal and allow quick recovery, program modification, and execution. All program debugging and modification operations are performed directly at the terminal in the MUMPS language.

Most MUMPS commands or functions can be executed from the keyboard in direct mode. When a command is entered, the MUMPS language interpreter immediately executes it and gives the appropriate response to the programmer. A command line can consist of several Standard MUMPS commands and arguments, comments, and data. For example, the programmer can enter the command line:

```
> WRITE "7+5=",7+5
```

This command tells DSM-11 to print the characters "7+5=" on the terminal, evaluate the arithmetic expression 7+5 and print the result on the terminal. DSM-11 responds by typing

```
7+5=12
>
```

The DO command tells DSM-11 to begin executing at a specified label of the stored program. It will continue until it encounters a control command such as GOTO or QUIT, or arrives at a point where there is nothing else to interpret.

To create a program, the programmer enters one or more lines of MUMPS commands. Once a program has been created, the programmer can store the contents of the partition's program buffer on disk. The program can then be reloaded into the program buffer from the disk. When a program is loaded in the program buffer, it can be modified by adding new lines or by replacing, deleting, or modifying existing lines of code.

Elements of the MUMPS Language

An expression is a value description that can be made in the Standard MUMPS language, including any legal combinations of operands and operators. The following are examples of expression elements:

123.34	constant
ABC	simple variable
"ABCD"	literal
MX(5)	local subscripted variable
^XYZ(2,5)	global variable
$LENGTH(Z)	function reference
(A+@B-(C/D))	subexpression

The operators in an expression serve to represent the various arithmetic and logical computations of the Standard MUMPS language. Following is a list of Standard MUMPS expression operators:

TYPE	SYMBOL FUNCTION
Arithmetic	+ Addition
	− Subtraction or Unary minus
	* Multiplication
	/ Division
	# Modulo
	\ Integer divide
Relational	< Less than
	> Greater than
	= Equality
Boolean	& AND
	! OR
	' NOT

String	[Contains] Follows
Relational	? Pattern verification = Equality
String	– Concatenation
Indirection	@ Indirection

Indirection is denoted by the character @ followed by an atomic expression. The value of the expression is substituted for the occurrence of indirection before the rest of the line is interpreted. Of special importance are the relational string operators. They provide facilities for determining the characteristics of string data.

The operators return true or false results. They are

- String Contains ([) — The string specified by the left operand is examined for the occurrence of the string specified by the right operand.
- String Follows (]) — The string specified by the left operand is compared character-for-character with the string specified by the right operand to establish relative position according to the ASCII collating sequence.
- Pattern Verification (?) — The string specified by the left operand is examined for the occurrence of the character patterns specified by pattern specification codes.

A command is the basic unit of expression in the Standard MUMPS language. A command is a mnemonic that symbolizes the action to be performed, such as GOTO or SET. The command name can be abbreviated to one letter. It usually takes one or more arguments that specify the objects of the action to be performed. Several Standard MUMPS commands can be present on a command line.

An optional Boolean-valued expression preceded by a colon can be used as part of an argument to specify conditional execution. For example, "GOTO LOOP:A>B" means that control is transferred to "LOOP" if A is greater than B.

The following is a list of DSM commands:

BREAK	Suspends execution of a routine and brings the terminal back to direct mode.
CLOSE	Releases one or more designated devices from ownership.
DO	Initiates execution of DSM routine at the label specified, with an implied return.
ELSE	Conditionally executes the statements following it.

FOR	Produces looping by repeating commands residing on the same line for a specific set of variable values.
GOTO	Interpreter execution is transferred either to a specified line or routine.
HALT	Ends your use of DSM-11.
HANG	Suspends program execution for a specified time interval.
IF	Permits the conditional execution of the commands or statement that follow it.
JOB	Starts a specified routine in a new partition.
KILL	Deletes the specified local and global variables.
LOCK	Makes a particular variable or node of a variable unavailable for locking by another user.
NEW	Saves named local variables, restores them at the next QUIT.
OPEN	Obtains ownership of one or more devices.
QUIT	Terminates the current flow of execution.
READ	Receives data from the current device.
SET	Assigns the value of an expression to a variable.
USE	Designates a specific open device as the current device for input and output.
VIEW	Allows you to read and write data to disk storage or to alter locations in memory.
WRITE	Sends data and/or control information to the current device.
XECUTE	Executes DSM-11 statements that result from the evaluation of an expression.

The following commands, known as Z-commands, are the DSM-11 extensions to the Standard MUMPS language:

ZALLOCATE	Allocates specified variables.
ZBREAK	Turns on, turns off, and controls the DSM-11 debugger.
ZDEALLOCATE	Deallocates all variables previously locked with ZALLOCATE.
ZGO	Resumes execution of a routine after a BREAK command.

ZINSERT	Inserts a line into the routine currently in memory.
ZJOB	Starts a specified routine in a new partition.
ZLOAD	Loads a routine into memory.
ZPRINT	Writes the current routine to the current output device.
ZQUIT	Directs control to the previously declared error-handling routine.
ZREMOVE	Deletes the current routine or specified lines in the current routine.
ZSAVE	Stores a routine in your routine directory.
ZTRAP	Forces an error, preventing routine from executing any further.
ZUSE	Allows temporary use of a terminal device owned by another job.
ZWRITE	Writes all local variables to the current output device.

A function performs an operation and returns a value, based on the outcome of that operation. The following is a list of available functions:

$ASCII	Returns the ASCII code of a string character as a decimal integer.
$CHAR	Translates a decimal integer into an ASCII character.
$DATA	Returns an integer indicating whether a specified node contains data, or has descendants.
$EXTRACT	Returns a substring of a string expression, selected by position number.
$FIND	Returns an integer specifying the end position of a specified substring.
$JUSTIFY	Returns a string, right-justified in a field of a specified length.
$LENGTH	Returns number of characters in a string.
$NEXT	Returns the subscript of the next sibling in collating sequence to the specified global or local node.
$ORDER	Returns the subscript of the next sibling in collating sequence of a specified array node.
$PIECE	Returns a substring from a specified string selected by delimiter.
$RANDOM	Returns a pseudorandom integer uniformly distributed in a closed interval.

$SELECT	Returns the value of the first expression in its argument list whose matched truth value expression is true.
$TEXT	Returns the specified line from the routine currently in memory.
$VIEW	Returns an integer between 0 and 65535, equal to the contents of the memory location specified in the argument.

Certain functions, called $Z-functions, are DSM-11 specific. They are provided as extensions to Standard MUMPS, giving more options to the user.

$ZCALL	Provides a general purpose function call to user-written routines.
$ZNEXT	Performs a physical scan of a global array.
$ZSORT	Returns the subscript of the next sibling in string collating sequence of a specified array node.
$ZUCI	Returns the UCI number, given the UCI name; or returns the UCI name, given the UCI number.

A number of special reference-only variables are defined within the system to control the flow of information and to provide system information to Standard MUMPS programmers. These variables are maintained and updated by the system for each job partition. The following is a list of the special variables, including the $Z special variables.

$HOROLOG	Contains the current date and time.
$IO	Identifies the current I/O device.
$JOB	Contains the job number.
$STORAGE	Contains the amount of free space available within the current partition.
$TEST	Contains a truth value computed from execution of the most recent IF command, containing an argument, or an OPEN, LOCK, or READ with a timeout.
$X	Contains a nonnegative integer value equal to the next column position to be output.
$Y	Contains the current line number.
$ZA	Contains status or error information for the current device.
$ZB	Contains status information on the current device, in the form of a numeric value.

$ZERROR	When an error occurs, this variable contains the line segment that caused the error.
$ZNAME	Contains the name of the current routine.
$ZTRAP	Contains a reference to a line and/or routine to which you want control to pass in the event of an error.
$ZVERSION	Contains the name and version of your DSM system.

Chapter 7 ▪ MicroPower/Pascal

▪ MicroPower/Pascal Has The Tools You Need for Realtime Applications

MicroPower/Pascal is an advanced software tool kit for developing PDP-11 (Q-bus) based microcomputer applications. It includes a high-performance, optimizing Pascal compiler, a modular executive, and all the tools you need to create concurrent, realtime application programs. You create these applications on a PDP-11 or VAX/VMS host system for execution and debugging in a separate target microcomputer that can be any Digital Q-bus processor from the FALCON SBC-11/21 to the PDP-11/23-PLUS. Each application is constructed especially for its target system, with the exact set of operating system services needed.

At present there are four MicroPower/Pascal products — MicroPower/Pascal-RT, MicroPower/Pascal-RSX, MicroPower/Pascal-Micro/RSX, and MicroPower/Pascal-VMS. MicroPower/Pascal-RT, MicroPower/Pascal-RSX, and MicroPower/Pascal-Micro/RSX develop applications using a PDP-11 or MicroPDP-11 host system, while MicroPower/Pascal-VMS is used for application development on the VAX family. Given the same set of inputs, all four products produce identical target runtime applications.

MicroPower/Pascal is particularly suited for dedicated, realtime microcomputer applications such as process control, instrumentation, and robotics.

Features of MicroPower/Pascal include

- Pascal language with extensions that support concurrent realtime programming.
- Symbolic debugger (PASDBG) to aid debugging of application programs running on the target runtime system.
- Modular Runtime System Software with shareable language interfaces for both Pascal and MACRO-11. Includes RT-11 compatible file system, communications service, selected device handlers, and executive service routines.
- All components required to support application development on a host development system are provided.
- Supporting the development of applications where the application software will be entirely ROM/RAM resident in the target runtime system.

Transporting an application to the target system can be done by

- Downline-loading via serial line interface.
- Programming PROM chips and transferring them to the target system (PROM programming hardware and software are not included with this product).
- Manually transferring a bootable application via a TU58 DECtape II cartridge, RX50, RD51, RX02 diskette, or RL01/RL02 disk cartridge.

The applications software is created and linked with the appropriate MicroPower/Pascal runtime software components on the host development system. MicroPower/Pascal provides the host development system software components needed for this step. When the application software is ready for debugging and testing, it is transported to the target runtime system. The application software can then be executed in the target runtime system. If a serial line is connected between the host development system and the console port of the target runtime system, the execution of the application software in the target can be controlled and tracked from the host with the help of the debugging tools provided in MicroPower/Pascal. The separation of the host development system from the target runtime system allows the user to make use of a high-performance host development system and test the application software in the target runtime environment. Figure 7-1 illustrates the host development and target runtime system components.

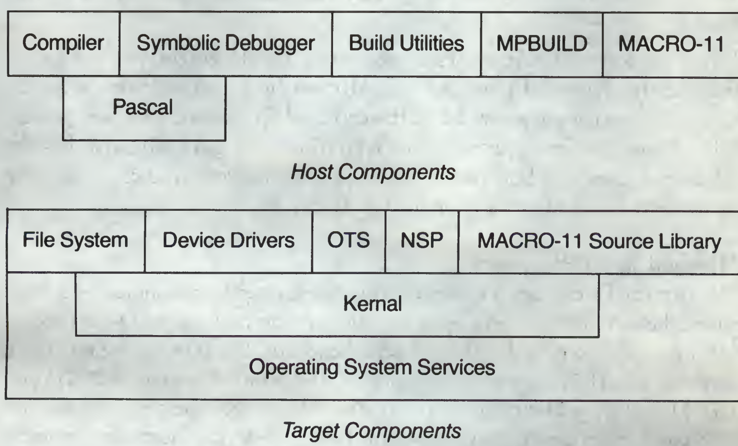


Figure 7-1 ▪ MicroPower/Pascal Host and Target Components

An extended version of Pascal is provided as the system implementation language and is suitable for most user applications. The extensions in the language enable the user to write realtime applications entirely in Pascal. However, MACRO-11 can also be used as the implementation language for sections of code. MicroPower/Pascal-RT is supported only with included RT-11 software.

MicroPower/Pascal-RT is a packaged software system that includes a subset of the RT-11 Operating System for use on the host development system. The RT-11 components include Extended Memory (XM) monitor, SYSMAC, SYSLIB, HELP, EDIT, KED, KEX and K52, MACRO-11/CREF, Linker (LINK), Peripheral Interchange Programs (PIP), Resource (RESORC), Librarian (LIBR), Device Utility Program (DUP), Directory (DIR), Queue Packages, DUMP, SRCCOM, BINCOM, FILEX, and FORMAT. The patching components include SIPP, SLP, and PAT.

▪ MicroPower/Pascal Supports Multitasking

MicroPower/Pascal is a microcomputer software architecture that extends standard Pascal to incorporate system implementation language capabilities and to support concurrent programming (multitasking). MicroPower/Pascal lets you code microcomputer applications that have direct access device registers, and can perform interrupt handling. Built-in procedures allow easy access to the file system, clock process, device drivers, and the operating system services of the kernel.

When you team MicroPower/Pascal with your PDP-11 system, you will have a complete hardware/software package with everything you need to build quality microcomputer applications. MicroPower/Pascal builds a complete and powerful realtime software application that is so compact it can reside in as little as 8 Kbytes of memory. For your most complex applications, you can address as many as 4 Mbytes of memory on the LSI-11/23.

Host and Target Processors

MicroPower/Pascal uses a two processor development environment — a host, where the compiler and development utilities reside, and a target Q-bus processor, where the compiled code and kernel execute. The host can be a PDP-11 running the RT-11 extended memory (XM) operating system, RSX-11M or RSX-11M-PLUS, a MicroPDP-11 running the Micro/RSX operating system, or a VAX computer running the VMS system. This provides the most effective work environment for developing target system programs. You can transport your final application program or the target microcomputer by one of three methods — writing it into read-only memory (ROM), downline-loading it over a serial line, or recording it on a magnetic storage medium such as a floppy disk or tape cartridge that you can bootstrap on the target.

Concurrent Programming Capability
Concurrent programming means your Pascal source code is structured into independent parts called processes that appear to execute simultaneously. Each process competes with all other processes for control of the target processor, but cooperates with all other processes in manipulating shared resources such as memory and peripheral devices.

Target System Kernel
MicroPower/Pascal processes have no need for a conventional operating system. Instead, every application contains its own customized set of routines, the kernel, that supports them. MicroPower/Pascal automatically selects those operating system services that your application requires from a library in the PDP-11 host computer and places them in a kernel. The kernel and the application it supports make the efficient use of memory requiring only system services.

The target system kernel lets applications access device registers, the file system, the clock process, and device drivers. The kernel also performs interrupt handling and lets concurrent tasks share the target processor.

Synchronizing Processes
Crucial to this concurrent application design is the mechanism for sharing control of the CPU and other common resources, such as data areas and peripheral devices. In the MicroPower/Pascal target system, executing processes are synchronized by *semaphores*.

Semaphores are global data structures that are manipulated by two or more processes. They act as flags that are raised and lowered by processes to signal their progress to other processes. You create semaphores in a source program to guide the response of the entire application to external, realtime events. Semaphores can delay a process until another process sends it a signal to proceed. This lets two processes share access to common data without corrupting the data.

▪ MicroPower/Pascal Host Components

MicroPower/Pascal utility programs construct your application and load it into the target system memory. They accept as input object modules of compiled or assembled source code contained in files. The utilities also accept object modules from various module libraries. They link these object modules with a customized kernel to create the application.

MicroPower's modular architecture makes it easy to test, debug, update, and expand your applications. It also reduces memory requirements for applications without sacrificing the ability to build large, versatile, 4-Mbyte configurations.

The MicroPower/Pascal utilities include

- MERGE
- RELOC
- MIB
- COPYB
- MPBUILD
- Automatic installation procedure

MERGE combines user-developed object modules into a single object module that resolves intermodule references. MERGE accepts multiple object modules containing compiled or assembled source code and data as input. It automatically combines program sections (p-sects) with identical names from all input object modules. MERGE also uses the symbol tables created in each object module during compilation to resolve intermodule references. For every reference to a declared EXTERNAL name, MERGE looks for a declared GLOBAL definition in the other object modules. Undefined symbols are flagged.

MERGE can be used to build a customized kernel that contains *only* the system services issued by the application program. The kernel is not relocatable; it must start at zero.

RELOC is a utility program that relocates merged object modules to specific virtual memory addresses. It produces a process image module with one base address. This base address is the module's correct location within the application's memory range. With RELOC, you can directly specify the virtual base addresses of different parts of the application. RELOC also separates p-sects according to their Read-Only or Read/Write attributes, and modifies the code sections to execute properly at their assigned addresses.

The *Memory Image Builder (MIB)* utility creates memory image files for execution on the target system. These files can be booted, downline loaded, loaded by the Pascal Debugger with an optional symbol file, or used for PROM blasting. The MIB utility creates the executable application by placing all its components into one structure, called the memory image. This memory image includes each merged, relocated piece of the application with intermodule references resolved. The MIB utility lets you control the placement of pieces of the application in memory.

COPYB writes a bootstrap for loading application images from a mass-storage device on the target runtime system. Currently supported runtime mass-storage devices include the TU58 DECtape II cartridge, RX02 dual-drive floppy disk system (both single- and double-density RX02 diskettes are supported), RX50 5.25-inch floppy diskettes, RD51, and RL01/RL02 cartridge disk systems.

MPBUILD automatically generates command files to run COMPILER, MERGE, RELOC, and MIB utilities.

MicroPower/Pascal Compiler

The MicroPower/Pascal compiler supports a superset of the Pascal language plus realtime extensions. The compiler runs on any PDP-11 with the RT-11XM, RSX-11M, RSX-11M-PLUS, or MicroRSX operating system and at least 128 Kbytes of memory. It also runs on,a VAX/VMS system.

The compiler contains a sophisticated global optimizer. It inspects all program modules for redundancy and produces code that executes almost as quickly as MACRO for a ROM and/or RAM environment. It significantly reduces the size and improves the speed of application programs by efficiently using the LSI-11's general purpose registers and hardware stack capabilities.

The compiler provides for separate module compilation and efficient interfacing to MACRO-11 assembly language routines. High-level language access to the modular runtime routines (kernel) means more efficient, and less expensive, system-level software development with little need for MACRO coding. You do not have to get involved with intricate machine instructions, nor do you have to learn separate operating system functions.

Compliance with International Standards Organization (ISO) specifications allows using the source code from other ISO compilers with full compatibility.

An extensive library of Object Time System (OTS) routines provides the compiler with runtime support for Pascal functions including utility and I/O routines and arithmetic routines such as floating-point support.

Symbolic Debugger

The Symbolic Debugger program, PASDBG, residing in the host system, downline-loads and remotely controls the execution of applications in the target processor. This is accomplished without expensive incircuit emulation hardware. Not only does it allow full program debugging in the original Pascal language terms, but it also enables the programmer to view the target's kernel concurrently executing processes. Debugging the application in the actual target system assures a more reliable final product.

Although PASDBG resides in the host, debugging an application program requires some additional code in the target. Building an application for debugging adds about 800 words to the size of the application. Setting the debug switch also increases the size of the code generated when compiling a module. However, you set the debug switch only on modules you are debugging. You add undebugged modules to the application a few at a time, debugging each, then rebuilding without debugger support. In this manner, the entire application is built from small, debugged pieces.

MicroPower/Pascal Target Components

The target environment includes an assembly language interface that allows MACRO-11 programming, device drivers that support many Q-bus interfaces, and a Runtime System Software.

MACRO-11 Source Libraries

The MACRO-11 interface to the runtime system is included in the form of a macro library, which is useful in developing MACRO-11 programs. Using MACRO-11 modules lets you handle the most time-critical applications.

Device and File Support

MicroPower/Pascal provides precompiled driver processes that act as interfaces between your application and various Digital devices (such as the RX02 floppy disk drive). You can also include a file system process and modules that allow you to create, access, and maintain data on target mass-storage devices in a format compatible with RT-11. Because driver processes and the file system are both fully accessible from your Pascal source code, I/O operations are much easier.

Runtime System Software

The Runtime System Software consists of a kernel and system processes included in the kit in the form of object libraries. The kernel is a modular executive that supports the following target system features — process synchronization, interprocess communication and scheduling, exception handling, interrupt handling, timer services, and memory management. The system processes consist of device drivers, a compatible file system, and communications service. The communications service comprises DECnet end node support on Ethernet and asynchronous serial lines, as well as MicroPower point-to-point on synchronous serial lines. The MACRO-11 interface to the Runtime System Software is included in the form of a macro library. Also included is an extensive library of Object Time System (OTS) routines that provide the Pascal compiler above with runtime support for Pascal functions and arithmetic routines, including floating-point support, utility, I/O, and math routines.

Chapter 8 • ULTRIX-11

ULTRIX-11 — A Native UNIX Operating System for Digital's PDP-11 Operating System

ULTRIX-11 Version 3.0 is an enhanced version of the AT&T Bell Laboratories UNIX operating system, Version 7. The UNIX general purpose, interactive, timesharing operating system was developed at Bell Laboratories in the late 1960s—largely on Digital's PDP-11 systems—to provide a flexible, elegant programming environment.

Originally designed by programmers for programming, the UNIX operating system was written to optimize application developers' efforts. One of the guiding principles behind the development of the UNIX operating system was that the software should optimize the programmer's time, not the computer's time. Over the years, the UNIX operating system has gained a reputation as a powerful, hardware-independent system. The level of compatibility between the different versions of the operating system determine how easy it is to take an application written under one UNIX operating system and move it to another UNIX operating system implementation.

ULTRIX-11 is the 16-bit version of Digital's ULTRIX family of UNIX-based operating systems. The ULTRIX family takes the best features of the UNIX operating system, and then adds the enhancements that optimize the software so that it takes advantage of the computer's design. With ULTRIX-11 Version 3.0, you get the best of both worlds — compatibility with AT&T's popular System 5.2, Release 2, plus the networking support and online help characteristic of Digital's operating systems.

ULTRIX-11 Version 3.0 Builds on AT&T's UNIX Version 7

As a native UNIX operating system for PDP-11s, ULTRIX-11 offers the programmer productivity features you expect from a UNIX-based operating system. But it also provides compatibility with other UNIX systems and features facilities that let you take advantage of Digital's traditional networking strength. (The end of this chapter provides summary information about the compatibilities between ULTRIX-11 Version 3.0 and other UNIX operating system implementations.)

A true UNIX operating system, ULTRIX-11 provides the following features:

- Source Code Control System (SCCS) for improving control over files during development. (ULTRIX-11 implements both the System III SCCS and the 4.2 BSD SCCS interface program.)
- Plot library, including support for the VT240 and VT241 in 4014 emulation mode.
- Choice of editors, including vi, ed, sed, and ex.
- C, Assembler, and FORTRAN 77.
- Standard I/O line buffering.

The ULTRIX-11 system enhances the Version 7 base with

- Source-level compatibility with AT&T UNIX System 5.2, Release 2, as defined in AT&T's System V Interface Definition.
- TCP/IP networking.
- Access to DECnet through an ULTRIX-32 or ULTRIX-32m node running DECnet-ULTRIX.
- Improved system performance based on an increase (from 512 to 1,024 bytes) in the amount of data written to or read from a disk per access.
- Online help facility.
- Automated system generation, system installation, and initial setup.

The ULTRIX-11 operating system supports a wide range of processors and peripheral devices (refer to the *ULTRIX-11 V3 Software Product Description*). It also features the new Digital Storage Architecture (DSA) disks, which use the Mass Storage Control Protocol (MSCP) and which are compatible with any other disk conforming to MSCP specifications.

▪ ULTRIX-11 Facilitates Communication and Networking

ULTRIX-11's impressive set of communication and networking facilities lets the ULTRIX-11 system become an integral part of your computing environment. The system provides access to Digital's powerful DECnet networks and supplies the most popular UNIX communications facilities.

ULTRIX-11 systems gain access to DECnet via an ULTRIX-32 or ULTRIX-32m node running DECnet-ULTRIX. With the DECnet-ULTRIX node providing the gateway, access to the network is transparent to the user. The DECnet access facility supports electronic mail, remote access to DECnet systems (remote login), and file transfer. (Users on an ULTRIX-11 systems can perform a remote login to the DECnet system; users on the DECnet system cannot set host to the ULTRIX-11 system.) The ULTRIX-11 system initiates file transfers to and from the DECnet system. The file transfer facility does not support wildcards in file names, and source and destination files must be given explicitly. Supported commands include mail, dlogin, dcp, dcat, dls, and drm. The ULTRIX-11 system reaches the DECnet host using a TCP/IP Ethernet connection to the ULTRIX-32 or 32m node.

TCP/IP allows the system to connect to other ULTRIX systems over an Ethernet using a DEQNA/DEUNA Ethernet. It also enables ULTRIX-11 systems to connect to other UNIX-based systems. Major TCP/IP networking capabilities include sending/receiving mail, remote access to other systems, and file transfer.

ULTRIX-11 also offers UNIX-to-UNIX Communication Protocol (UUCP), which is a series of programs designed to allow communication among ULTRIX family systems or any other UNIX-based UUCP system using the "g" protocol. With UUCP, you can transfer files, and remote commands can be executed via dialup or hard-wired communication lines. These files are created in a spool directory for processing by the UUCP daemons (background processes) and executed in batch mode.

Additional facilities include

- Mail, which allows users to transmit text and data files, and define aliases and message forwarding instructions. The system also provides notification of new mail.
- Tip, which establishes a full duplex-connection to another processor. Tip provides virtual terminal access to the remote machine and file transfer between ULTRIX and UNIX-based systems. This utility supersedes the CU (call UNIX) command, but supports the CU user interface for compatibility.
- Tar, tp, and cpio utilities allow users to save and restore individual files or selected directory subtrees on tape or RX50 diskette. Tar can also save/restore empty directories and special files.

ULTRIX-11 Features Automatic System Generation

Generating an ULTRIX-11 system is easy with the interactive system generation program, SYSGEN. The SYSGEN program features

- Interactive sysgen, with online help.
- Maximum of four user-written device drivers containing kernel linkages for a maximum of four users. A commentary on writing device drivers and a sample device driver are also provided.
- Interactive configuration of devices.
- Interactive specification of kernel size/tuning parameters.
- Automated build of operating system kernel.
- Memory-resident overlays (up to 15 8-Kbyte overlays) for expanding kernel text space on both split I/D space and nonsplit I/D space processors.
- In-kernel floating-point hardware simulator.
- ULTRIX-11 System Acceptance Test (USAT). The USAT verifies installation of the operating system, tests operation of major subsystems, and tests more than 125 commands and minor subsystems.

The SYSGEN program can also list, print, and remove configuration files and can print a list of the devices supported by ULTRIX-11, along with their ULTRIX-11 device mnemonics. SYSGEN includes extensive online help to guide you through the generation process.

The ULTRIX-11 System Acceptance Test (USAT) lets users verify the installation and proper functioning of all major subsystems.

ULTRIX-11 Offers a Choice of User Interfaces

ULTRIX-11 offers you a selection of command language interfaces — the UNIX Version 7 Bourne Shell, the System V Bourne Shell, and the C Shell (including job control for switching between processes running in the foreground and background). As with all UNIX systems, these shells provide a tailorable command language interface to the system. The System V or ULTRIX-11 programming environment is selectable on a per-user basis.

The shells let you combine commands to create your own custom commands. The shells can produce their own command lines and command files that

- Assign symbolic names.
- Evaluate numeric and logical expressions.
- Accept parameters.
- Communicate with interactive users invoking the shell script.
- Perform conditional and branching logic.

Two terminal drivers are available — the standard UNIX Version 7 terminal driver and the newer Berkeley 2.9 BSD terminal driver — and are selectable on a per-user basis. Users can also set the terminal control characters, thus facilitating "erase" and "kill" processing for videoterminals. ULTRIX-11 supports the standard Version 7 TTY driver and the newer Berkeley 2.9 BSD TTY driver that is used with job control.

▪ Select the Editor Best-suited to Your Needs

ULTRIX-11 offers a full complement of UNIX editors that let you edit, expand, copy, search, and delete text. Choose from

- vi, the full-screen editor.
- ed, the basic line-oriented editor.
- sed, the stream editor.
- ex, the extended line editor.

▪ ULTRIX-11 System Includes Error Logging and Improved Fault Tolerance

The ULTRIX-11 Operating System Error Logger collects information on system and device errors as they occur and records this data in the error log file for later analysis. The error logging system has been updated to include support for new devices supported by the ULTRIX-11 operating system, and consists of the following components

- The Error Log Initialization (ELI) program performs housekeeping functions, including saving the error log file, zeroing the error log file, enabling and disabling error logging, and printing the size of the error log files.
- The ULTRIX-11 operating system kernel and device drivers have been modified to gather information and save it in the kernel error log buffer.

- The Error Log Copy (ELC) process is a background process that copies error log records from the kernel buffer to the error log file.
- The Error Log Print (ELP) program formats the error log data and generates error reports.

The ULTRIX-11 operating system also provides improved fault tolerance, and supplies more complete error information when a fault does occur. All operating system and device error messages have been documented.

ULTRIX-11 Offers Bad Block Replacement

The ULTRIX-11 operating system implements a bad block replacement strategy for RK06/7, RM02/3/5, and RP04/5/6 disks. The drivers for these disks have been modified to read the bad block file and automatically replace any bad blocks. The replacement is transparent to the operating system and the users. The ULTRIX-11 operating system implements bad blocking with three stand-alone programs:

- Stand-alone Disk Bad Block Scan (BADS).
- Stand-alone Disk Initialization (DSKINIT).
- Stand-alone Static Bad Block Replacement (RABADS).

The RABADS program, in conjunction with the disk hardware, replaces bad blocks on MSCP-type disks (RD51, RD52, RD53, RA60, RA80, RA81, and RC25). It also includes a Bad Block Status Command (BADSTAT) that monitors and displays the number of replacements performed on each bad block on RK06/7, RM02/3/5, and RP04/5/6 disks. This command enables system managers to assess the effect bad block replacement has on system performance.

▪ ULTRIX-11 Offers a Variety of Tools for Software Development

A summary of the various ULTRIX-11 operating system software development tools is provided below. These facilities include

- Languages
  - C compiler from AT&T System V.
  - V7 C compiler as a backup.
  - Portable C compiler (pcc).
  - FORTRAN-77 plus structure and beautify utilities.
  - Pascal from Berkeley 2.9 BSD.
  - UNIX assembler.
  - MACRO-11 assembler, linker, and cross-reference utility.
  - awk pattern scanning and processing language.
- Loader
  - Links object code to make it executable.
  - Overlay load option allows creation of large text user programs. For split I/D processors the maximum program size is 408 Kbytes (maximum 56-Kbyte data space and 8-Kbyte stack). For nonsplit I/D processors the maximum program size is 208 Kbytes (maximum 40-Kbyte data space and 8-Kbyte stack).
- Archiver
  - AR creates and manages libraries and archives.
- Debuggers
  - ADB object and executable level debugger.
  - CTRACE, C program debugger.
- Tools
  - Source Code Control System (SCCS)
  - C program portability checker (lint)
  - Lexical Analyzer Generator (lex)
  - Parser generator (yacc)

▪ Graphics, Online Documentation and Help, and User Login Limits

ULTRIX-11 Version 3 features graphics, online documentation and help facility. Also featured is a change in the user login limits.

Graphics

The UNIX V7 plot library has been modified to support Digital VT240 and VT241 terminals in Tektronix® 4014 emulation mode. The plot library also supports the Digital LA50 and LA100 printers, Digital ReGIS output devices, and the Digital GIGI terminal.

Online Documentation and Help

The ULTRIX-11 Operating System includes the General System Help Facility that includes menus showing how to use the help facility and general use commands including common syntax, a brief programming environment that is selectable on a per-user basis.

The terminal driver is selectable on a per-user basis with the stty command. Two terminal drivers are available — the standard UNIX V7 terminal driver or the newer Berkeley 2.9 BSD terminal driver. Terminal control characters can also be set with the stty command. The ULTRIX-11 Operating System also includes the terminal capabilities database (termcap).

User Login Limits

The number of concurrent user logins is limited to 16 or 32 by Digital's binary sublicensing agreement with AT&T. The ULTRIX-11 Operating System Version 2.0 enforced this restriction by limiting the number of terminals enabled for logins (gettys running) to 16 or 32. The ULTRIX-11 Operating System Version 3.0 is less restrictive. It allows up to 100 terminals enabled for user logins, but limits the number of concurrent user logins.

ULTRIX-11 Version 3.0 Features Improved System Performance

With the addition of standard I/O line buffering and a new 1-Kbyte block file system, ULTRIX Version 3.0 provides performance significantly above that of ULTRIX-11 Version 2.0. These new features include

- *Standard I/O line buffering*—This change significantly improves the performance of any program using the standard I/O library routine by reducing system overhead. This change is most visible during output to a terminal at higher bit rates (1,200 b/sec or higher).
- *New 1-Kbyte file system*—By using a logical block size of 1,024 bytes instead of 512 bytes, the new file system effectively doubles file system throughput. The increased file system throughput improves overall system performance by providing faster access to data files and speeding up execution of all programs.
- *New UNIBUS map allocation scheme* (similar to Berkeley 2.9 BSD)—This scheme improves device I/O overlap and overall system I/O throughput on UNIBUS processors with 22-bit addressing (PDP-11/24 with KT24, PDP-11/44, 70, and 84).
- *Operating system kernel routine optimizations*—These optimizations speed up all memory-to-memory copy operations, produce faster path-name resolution, change the callout table to a linked list, reduce overhead significantly in a clock routine, reduce kernel overlay switching overhead by making as many functions as possible static, and reduce executive overhead (argument/environment passing).

ULTRIX-11 Provides Compatibility with Other UNIX-based Systems

ULTRIX-11 offers the compatibility with other UNIX system implementations that lets you share software developed on other systems. The following sections discuss the level of compatibility ULTRIX-11 shares with AT&T System V, AT&T System III, ULTRIX-11 Version 2.0, ULTRIX-32 and ULTRIX-32m, Berkeley 2.8 and 2.9 BSD, and UNIX Version 7.

ULTRIX-11 Compatibility with AT&T System V

The ULTRIX-11 operating system is compatible with System 5.2, Release 2, at the source code level. This source code level compatibility allows applications written for the System V environment to be compiled, linked, and then run on the ULTRIX-11 system. ULTRIX-11's system compatibility with System V is based on AT&T's System V Interface Definition Spring 1985, Issue I. Table 8-1 shows the degree of compliance, based on the sections defined in AT&T's Interface Definition.

Table 8-1 ▪ ULTRIX-11 Compliance with System V Interface Definition

Interface Definition Description	ULTRIX-11 Compliance
Section 2 (Base System)	
Operating system services	100%
Error conditions	100%
Signals	100%
Other library routines	100%
Header files	100%
Utilities	Not defined by AT&T
Environment variables	100%
System resident data files	100%
Directory tree structure	100%
Special device files	100%
Section 3	
Kernel extensions	100%
Section 4	
Basic utilities extension	100%
Advanced utilities extension	72%
Software development extension	75%
Network services extension	Not defined by AT&T
Large machine extension	24%
Graphics extension	Defined as obsolete by AT&T
Basic text processing extension	100%
User interface services extension	0% (no compliance)
Database manager extension	Not defined by AT&T

ULTRIX-11 Compatibility with AT&T System III

AT&T System III was released in 1982 as the first commercial UNIX product. The following summarizes ULTRIX-11's compatibility with System III.

- File systems are not compatible, but a migration path utility is provided.
- Bourne Shell scripts are syntax-compatible and upward compatible with the System V shell.
- C source code that was written for System III and upward compatible with System V is also compatible with ULTRIX-11.
- Objects and executable image formats are not compatible.

ULTRIX-11 Version 3.0 Compatibility with ULTRIX-11 Version 2.0

Digital's ULTRIX-11 Version 3.0 provides enhancements to its predecessor, ULTRIX-11 Version 2.0.

- File systems on Version 3.0 are not compatible with Version 2.0. However, ULTRIX Version 3.0 does provide a migration path utility.
- Bourne Shell and C Shell scripts are syntax-compatible.
- C source programs are compatible. The new (System V) C compiler has a stricter error checking facility. Many system calls and library routines have been added.
- Linking Version 2.0 object code is not compatible.
- Executable images are compatible.

ULTRIX-11 Compatibility with ULTRIX-32 and ULTRIX-32m

ULTRIX-32 is Digital's native UNIX operating system for VAX hardware; ULTRIX-32m is a version of the ULTRIX-32 software designed to run on a MicroVAX system. ULTRIX-32 software is an interactive timesharing operating system derived from VM/UNIX (Version 4.2 BSD) developed at the University of California at Berkeley. It supports VAX virtual memory architecture with demand paging, and features programmer productivity tools and a range of networking capabilities.

The following summarizes the compatibilities between ULTRIX-11 and the ULTRIX-32 products.

- File systems are not compatible.
- Bourne Shell and C Shell scripts are syntax-compatible.
- Source programs written in C that contain no architecural references will run on both the ULTRIX-11 system and the ULTRIX-32 products.
- Object code and executable images do not move between the ULTRIX-11 system and the ULTRIX-32 systems.

ULTRIX-11 Compatibility with Berkeley 2.8 and 2.9 BSD

Versions 2.8 and 2.9 BSD (Berkeley Standard Distribution) reflect early modifications developed by the University of California at Berkeley for the UNIX operating system. The following summarizes the compatibility between ULTRIX-11 and these BSD releases.

- Berkeley 2.8/2.9 BSD file systems are compatible with ULTRIX-11's file system. However, disk partition layouts are not compatible. ULTRIX-11 provides a migration path utility.
- Bourne Shell and C Shell scripts are syntax-compatible.
- C source programs are compatible.
- Linking of Version 2.8 and 2.9 BSD object codes is not compatible.
- Executable images run on all three systems.
- System calls from 2.9 BSD systems are compatible with those from ULTRIX-11.
- All 2.9 BSD PDP-11 library routines except loadav(3) and crypt(3) are compatible.
- Utilities under 2.9 BSD are compatible with ULTRIX-11 utilities.

ULTRIX-11 Compatibility with UNIX V7

The ULTRIX-11 system is derived from UNIX V7 and maintains a high degree of compatibility with this UNIX version. With the exceptions listed below, all of the UNIX V7 system is included in the ULTRIX-11 system.

- UNIX V7 file systems are not compatible. However, ULTRIX-11 provides a migration path utility.
- Bourne Shell scripts are syntax-compatible.
- C source programs are compatible.
- UNIX V7 object code does not link under ULTRIX-11.
- Executable images are compatible.

Chapter 9 • Interactive Application System (IAS)

▪ IAS Is a Proven Performer for a Variety of Applications

Digital continues to support IAS, a mature operating system used extensively in business, defense, and other government operations. IAS offers multiuser timesharing and supports concurrent interactive, batch, and realtime applications. IAS includes the MACRO assembler (bundled), SORT, the RMS record management system (bundled), a database management facility, and the FCS file control services. As an option, DATATRIEVE, an interactive query, report-writing, and data maintenance system, can be added. Several high-level languages such as FORTRAN IV, FORTRAN-77, PDP-11 COBOL, and BASIC-PLUS-2 can also be added. IAS features

- A single, easy-to-learn and easy-to-use interactive command language.
- Priority scheduling for realtime tasks.
- Submission of batch jobs from interactive terminals.
- Timesharing services for development of interactive applications programs.
- A simple internal software interface for the development and use of special-purpose, multiuser interactive applications.
- A sophisticated file system providing device independence; file protection; sequential, random, and relative file access; and, optionally, multikeyed ISAM.
- System management facilities for system configuration, generation, and control.
- Facilities to account for and restrict the use of system resources.
- Dynamic allocation of system resources.
- Use of shared, reentrant code to minimize memory requirements.

IAS supports a variety of peripherals useful in batch and realtime applications, including lineprinters, card readers, and laboratory peripherals.

As a batch system, IAS services multiple queues of batch jobs. FORTRAN, MACRO, BASIC-PLUS-2, and COBOL jobs can be submitted to batch. The user interface for batch processing is the same as the Program Development System (PDS) interactive interface. Therefore, programs can be developed in interactive mode and run in production in batch mode. The system manager controls the amount of service that batch jobs receive from the processor.

As a generalized, flexible base for executing interactive applications, IAS provides support for application-specific user interfaces for applications such as data entry, bank teller terminals, or engineering computation, where it is necessary or desirable to present a customized interface to terminal users (operators, for example).

Further, IAS supports the concurrent execution of multiple interactive applications. Thus, a data processing application and the program development system can execute concurrently and be serviced jointly by the timesharing facilities of the system.

The program development system, PDS, provides a computing environment that supports most application processing requirements of IAS users. As such, it presents to IAS terminal users a standard interface that requests and processes valid passwords and user names before making system facilities available. The interface allows the user to create programs, submit jobs to the batch stream, and issue commands to create and manipulate program and data files.

The interactive application facility is further enhanced by the capability of the FORTRAN IV compiler and IAS to develop and support shareable programs. For the user, this means that system overhead (memory occupancy and swapping time) is minimized. Also, the user can allocate specific application interfaces and deallocate them as required. This facility is flexible and extendable. The system is easily modified and additional applications are easily added.

Special purpose interfaces can be written and checked out using the IAS program development system and then installed by the system manager for use on specific terminals. IAS provides a number of system services that can be called from the application program to enhance the function of these special purpose interfaces.

IAS provides the realtime processing facilities of multiprogramming, priority scheduling, power/fail restart, contingency exits, disk-based operation, and task checkpointing of realtime tasks. Realtime, interactive, and batch operations can occur concurrently and, normally, in that order of priority.

IAS system operations are managed by two executives. The realtime executive schedules realtime activities according to their priorities and manages the system resources not allocated to the timesharing activities. The timesharing executive schedules timesharing users on the basis of a time-slicing algorithm when realtime activities do not take precedence. Batch processing normally uses processor time available after interactive users are serviced. Both batch tasks and interactive tasks run under control of the timesharing scheduler.

The IAS System Monitor Provides System Flexibility

The IAS operating system is controlled by a system monitor consisting of a realtime executive kernel and an optional timesharing scheduler. The primary functions of the kernel include memory and disk management, supervision of privileged tasks (including realtime tasks and device handlers), file management, and maintenance of the general integrity of the system. The kernel maintains the Active Task List (ATL) to control task dispatching.

The timesharing scheduler (a sysgen option) controls both interactive and batch processing. It manages the execution of timesharing tasks by timeslicing and by swapping tasks into and out of memory.

Active Task List

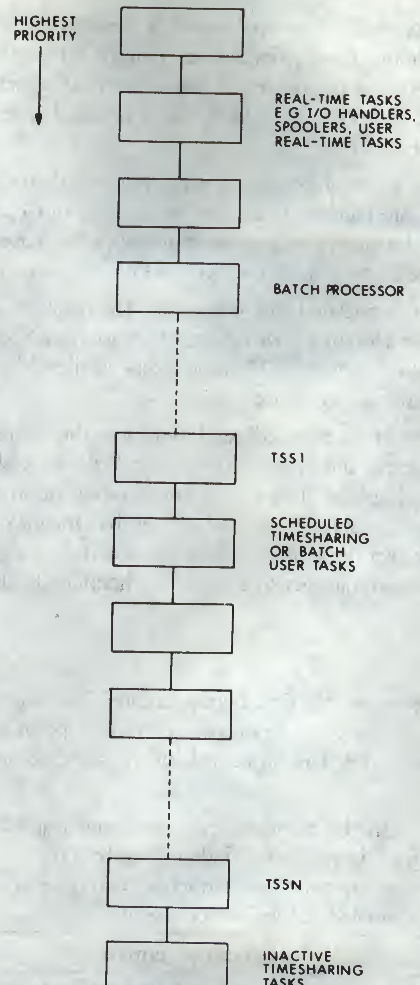
The kernel coordinates the dispatching of all tasks on the system by scanning the entries in the Active Task List (ATL), a priority-ordered list of all resident active tasks in the system. Because of their requirement for immediate service, the I/O device-handler tasks are put at the top of the ATL. For the same reason, any user-designated realtime tasks are assigned to high-priority levels. The timesharing scheduler, which runs at a lower priority than I/O and realtime tasks, controls the scheduling of user timesharing tasks by inserting tasks in the ATL. Figure 9-1 illustrates the priority structure of the ATL.

Three "pseudo-tasks," called TSS1, TSS2, and TSSN, are used to control the dispatching of tasks. The timesharing scheduler—TSS1—selects a task for execution by placing its entry in the ATL at a priority equal to itself. The scheduler then relinquishes control (for example, it waits for an event flag such as "time slice complete") to allow the kernel to dispatch to the user task. TSS2 is another pointer in the ATL.

TSSN is the null job: it runs continuously in a loop executing at priority 1, so that tasks below it on the ATL can never execute. When a timesharing task is not executing, TSS1 places the ATL entry for that task below that of TSSN.

Timesharing Scheduler

The objective of the scheduler is to reduce as far as possible the average response time to all user demands. In order to do so, it distinguishes between various levels of user importance and urgency of service. The scheduler maintains a number of queues, or levels, of tasks to be scheduled. It scans each level (high to low) in a round robin fashion until it finds a memory-resident runnable task. (A nonresident ready-to-run task will cause the swapping system to be activated.)



9-1 ▪ Schematic Diagram of ATL Structure

A task that uses a full-time quantum is transferred to the next lower level unless it is already at the lowest level. Tasks at lower levels are not scheduled as often as tasks at higher levels, but they are given a longer time quantum when next activated. The goal is that large jobs are run and swapped less frequently, but in compensation, receive more processor time once activated.

To prevent tasks from being starved of processor time because the scheduler is continuously scheduling higher priority tasks, a means of promoting tasks from one level to the level above is provided. If, over a given period of time, no scheduling has been performed at a given level, then a task at that level is moved to the bottom of the level above.

The scheduler finds a runnable task that is not resident, then the task must be loaded (swapped) into memory to receive its quantum of CPU time. Space is created in memory by moving resident tasks to create the required contiguous space, and, if necessary, by writing inactive tasks to the swap area on disk(s).

Two time factors are associated with every task. The quantum determines the amount of CPU time a job may have before it is swapped out of main memory. The time slice is the maximum CPU time a task is allowed to use before a rescheduling operation is performed.

The time slice parameter can be adjusted to achieve the desired compromise between responsiveness and system throughput. If the time slice is set to its maximum value, all tasks will execute without interruption for their entire quantum. The time slice should never be smaller than the maximum quantum for a Level 1 task. All the parameters of the scheduling algorithm can be adjusted by the system manager to tailor IAS scheduling to the needs of the installation.

Batch Processing

Batch runs as if it were another timesharing terminal. The batch command language is the same as the general-purpose, interactive, program development command language, and it is processed by the same command language interpreter.

Command input for the batch processor comes from a queue of commands. The batch queue itself is maintained independently, thus enabling jobs to be submitted at any time. The processor can service two types of queues. The system can maintain a spooled queue that consists of

- Batch job files submitted from interactive terminals.
- Command input from the card reader (if the card reader is designated as a spooled device).

Batch processing is initiated and terminated by the system manager, and the batch processor executes at the batch scheduling level, serviced by the timesharing scheduler. Though batch processing shares CPU time with interactive tasks, in timesharing systems its priority for service is always below that of the active tasks.

To assure that batch processing receives adequate service, the system manager can specify the percentage of CPU time to be made available to it, and the length of time (quantum) batch should run when it does receive service. For example, the system manager could direct IAS to devote ten percent of the available time to batch jobs in 2-second quanta.

Executive Data Structures

The IAS system maintains a number of common areas in which the various executive tasks store information and communicate with each other. SCOM contains the system tables, kernel node pool, lists, and some servicing routines. SYSRES, the System Resident Library, contains common routines that will be used by most tasks. IASCOM is a library containing timesharing nodes, lists, tables, and common routines for manipulating the timesharing data structures. IASBIF is a buffer area used for communication between the timesharing control primitives, IASCOM, and the timesharing executive.

I/O Services and Device Independence

Input and output constitute a significant part of all programmed activity. Thus, IAS provides a variety of services to perform these operations.

The IAS file system is a collection of system services that permits the user to view I/O as a transaction between a program and a named, protected collection of records known as a file. The file system manages all data transfers and provides the mechanism whereby a file intended for a record-oriented device, such as a lineprinter, can be dynamically directed to an area on magnetic storage.

Access to a user's files stored on a disk, DECtape, or labeled magnetic tape is controlled by a protection specification on each file. When creating a file, a user can specify whether other users may have access to the file and, if so, whether they may modify the file or merely read it.

One of the goals of any file system is to make the user program independent of the I/O hardware. Thus, while the storage characteristics of a medium are organized around physical records, the user deals only with logical records.

To provide greater device independence, the IAS user will in general use logical units instead of referring directly to physical devices. IAS provides a set of logical unit numbers (LUNs) that are not associated with specific physical devices or files until runtime. In the source program, all device and file references use LUNs. These LUNs may be assigned to particular devices by a command issued before the program is executed.

Sharing of Common Routines

In a system designed to support many users, there is a high probability that many tasks will use the same code sequences, such as mathematical routines and specialized I/O routines.

The common code could be built directly into each task requiring it, but this might result in several copies of the same code occupying memory space at the same time. The alternative employed by IAS is to put the (shared) common code where all users can share it, so that only one copy of the code is required.

Under IAS, shared areas may be either data areas (global common), sets of common routines (libraries), or the pure (read-only) areas of complete tasks (shared tasks). Global common areas allow simultaneously active tasks to share data. A shareable library consists of routines that may be interrupted to service another request, then resume execution later at the point of interruption. (Users who write reentrant routines can include their own shareable libraries in the IAS system.) Shared code does not need to be permanently resident; it can be loaded at the time a task that uses it is run. Programs written in either FORTRAN IV or MACRO can be shared.

▪ Command Language Interpreters Help You Communicate with Your System

A command language interpreter (CLI) is a task which manages the interface between a person and the IAS system. PDS, for example, is the program development CLI supplied with IAS. It allows the general user to access all nonprivileged facilities of the system. Another CLI called the system control interface (SCI) helps the system operator alter the state of the system, to designate user interfaces (CLIs), and to allocate facilities to each other.

While PDS is the standard command language interpreter to which a general terminal is allocated, the system operator can designate another specific task as the CLI for a terminal. For example, the operator might set aside one terminal to be used solely for program editing.

Users can write their own CLI tasks, which can be installed and allocated to timesharing terminals. Such user-written CLI tasks may define their own command language, which can be as simple and understandable as required, specifically designed for a particular application operation. Therefore, application terminal users do not have to learn a generalized command language such as PDS to perform their set of daily activities.

A CLI is written as a normal, nonprivileged task that can use, in addition to the standard system directives and file system facilities, the IAS system's timesharing control services. It can be written in any language that provides the facilities it requires; for example, a CLI that wishes to use the system QIO directive must be written in FORTRAN, MACRO, or BASIC (with user-defined functions).

The following two sections describe the two standard CLI tasks provided with the IAS system: PDS, the program development system, and SCI, the system control interface.

Program Development System (PDS)

A typical timesharing user interfaces with IAS through the program development system (PDS) command language interpreter. Under PDS, users can create, compile, link, load, and run programs. They can submit jobs to the batch stream, use various peripheral devices, and obtain system information.

PDS is a prompt-oriented system. After PDS is activated at a terminal, it invites the input of a command by issuing the prompt PDS. The user replies by typing a command name and its parameters, if any, followed by a carriage return. If a user does not supply all the parameters required in a command, the system will prompt for them. Additionally, the user can issue the HELP command to display the commands available.

The user can supply PDS commands in a file (called an "indirect file") rather than typing them in one at a time on the terminal. PDS processes the file in the same manner that it processes commands typed individually on the console. The commands, as well as any error messages that occur during the execution of the commands, will be displayed on the user's output device.

There are several types of PDS commands: some that provide access or system information, some that allocate resources, some that manipulate files, and some that control task execution. The system manager can, when defining user accounts, designate certain PDS commands as privileged or nonprivileged for any particular users, by specifying which commands a user can issue. For example, some PDS commands control realtime task execution, and only those users who have been given appropriate privileges can issue them.

Except for the LOGIN, LOGOUT, JOB, and EOJ commands, all nonprivileged commands can be issued in either interactive or batch mode. When a command is issued in batch mode, it requires a dollar sign ($) preceding the first character of the command name.

In addition to the general PDS commands, IAS includes special PDS commands available only to the system manager, and only when he or she is logged in under the system management account. There are three types of privileged PDS system managements commands:

- Accounting commands to authorize users and report system use.
- Realtime system control commands.
- Volume and file control commands.

System Control Interface (SCI)

System operators communicate with IAS through the system control interface (SCI) command language interpreter. The SCI command language uses the same syntax and conventions as the PDS command language, including prompting for missing parameters. Indirect SCI command files are also supported.

SCI commands enable an operator to monitor the system in four different areas. These areas are

- Command language interpreter control.
- Overall system and task control.
- Peripheral device control.
- System information.

- COMMAND LANGUAGE INTERPRETER CONTROL

The command language interpreter (CLI) commands allow the operator to install and remove CLI tasks, allocate and deallocate resources (e.g., terminals) to a CLI task, and abort a CLI task at a particular terminal. These commands are used both to initialize a timesharing system and to modify the system's characteristics during system operation.

- OVERALL SYSTEM AND TASK CONTROL

The system and task control commands enable the operator to load and unload device handlers that are not permanently resident; mount and dismount volumes; set the system parameters to suit the current workload; and shut down the system. These commands also enable the operator to have ultimate task execution control. For example, the operator can terminate any task in the system. This can be useful when, for example, a batch task loops indefinitely because of internal errors.

- PERIPHERAL DEVICE CONTROL

Peripheral device control commands provide the operator with the facility to service user requests for access to disk packs, magnetic tapes or other removable media. Additionally, the operator can control the output spooling mechanism and the type of printer forms being used.

- SYSTEM INFORMATION

The system information commands allow the system operator to display system information such as the active task list, CLI allocations, partition names and sizes, date and time, and device status.

SCI also allows all PDS commands.

▪ IAS Also Provides Special Tasks and Utilities

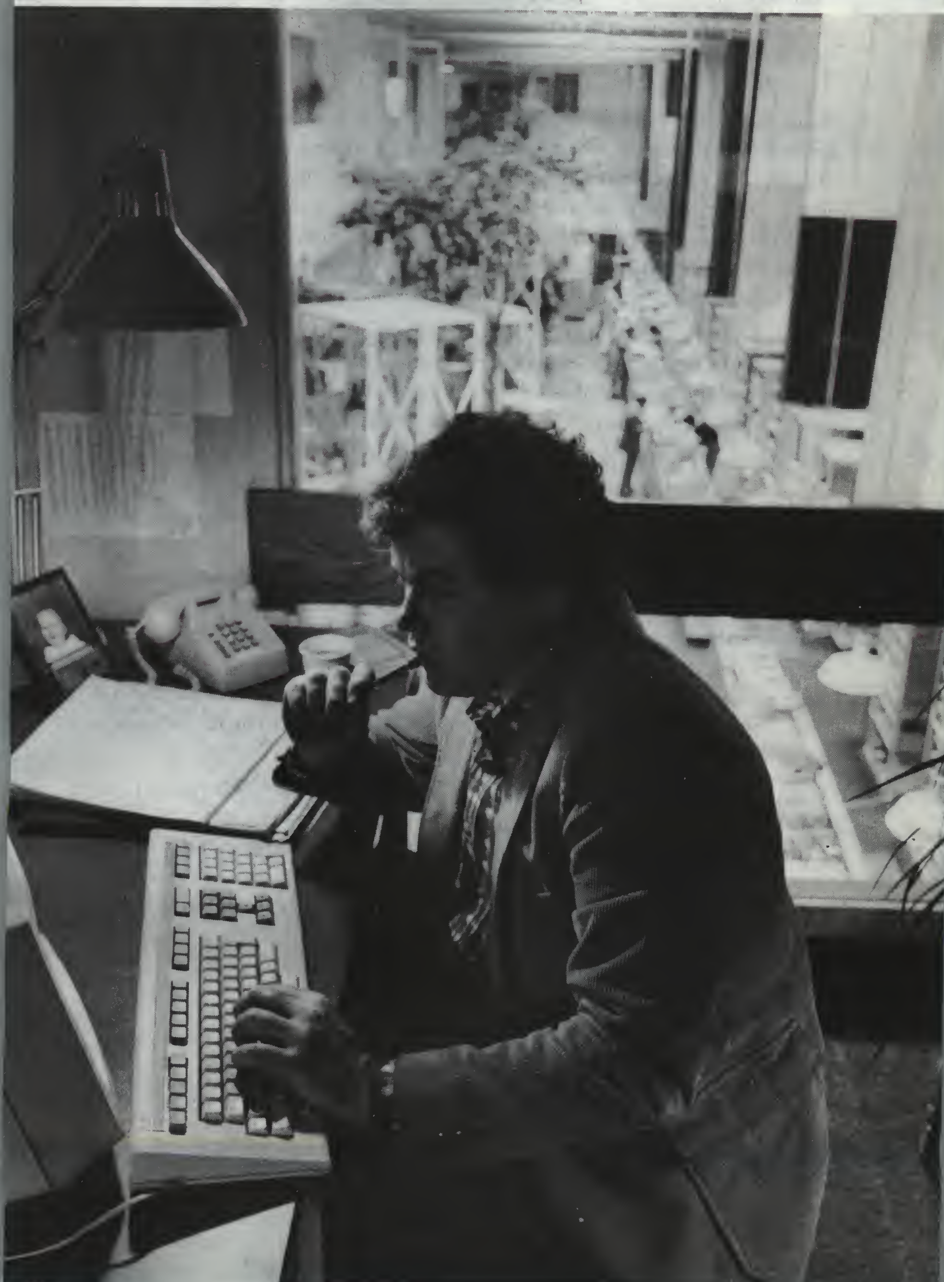
IAS provides a common command language for all standard system program development utilities such as the editor, linker, and librarian.

In addition to the standard program development utilities, IAS also provides two special system tasks called VERIFY and BAD BLOCKS. These tasks are available only to the system manager. VERIFY is used to verify the consistency and validity of the files on a Files-11 volume. BAD BLOCKS is used to locate any unusable blocks on a disk and is normally run prior to disk volume initialization.

The system manager or operator also has available a special utility called CDA (Core Dump Analyzer), a task that executes online with other tasks to capture system information at the time of a crash. It provides the capability to analyze the state of the system at the time the crash occurred.

General users have access to a special utility called PRESERVE. PRESERVE is a multiuser task that creates copies of disk, magnetic tape, or DECtape volumes. PRESERVE can also be booted into memory as a stand-alone program. Other utilities, such as DSC (disk store/compress) and BRU (a backup utility), are explained in Chapter 17, "File Management Utilities."

Chapter 10 ▪ Programming Languages



Programming Languages Make Your Computer Work for You

Doing useful work on a computer depends on the ease with which you can communicate your information and requests to it. Different circumstances determine different methods of programming, and broad categories of problems are often treated in different ways. This accounts for the growth of many different programming languages.

Some languages, such as FORTRAN, were originally intended for processing enormous amounts of numerical data through complicated formulas at high speeds. Others, such as COBOL, were developed for commercial applications in which there wasn't so much computing, but there was more data management. And still others, like BASIC, were invented to provide easy, nonthreatening access to students, so that they could quickly use the computer for problem solving, rather than worry about the intricacies of programming.

While some of these distinctions have become blurred over time, it is still true that certain kinds of problems are best attacked through certain kinds of languages, and the chapters that follow attempt to show the special strengths of each Digital-supplied language in satisfying specific application needs.

The American National Standards Institute (ANSI) defines standards for the various computer languages typically found in the United States. Such a standard is designated with a year suffix. FORTRAN-77, for example is the most recent FORTRAN standard; a programmer familiar with that language can tell immediately what general capabilities will be available in the language. Of course, computer languages are not frozen, and over time new standards develop either in industry or government, based upon need and pressure from users of the language. Consequently, most vendors offer standard languages with enhancements, so that they meet both government requirements and the needs of the programmers. Subsequent chapters explain Digital's enhancements to ANSI standards.

One of the great benefits of standardization is that each operating system takes care of the implementation of any particular language. For example, FORTRAN in the RSTS/E system might operate much differently from that in the RSX-11M system, but the programmer need not really know how those differences are managed. All that is needed to program with complete ease is to spend a day or two learning system-specific characteristics.

▪ Language Processors

There are three types of language processors in programming. What they all do is take the words and symbols typed by the programmer at a terminal and translate them into a series of binary codes that the computer can understand. When the computer is to output information, the processors take the binary codes and return them to a format that can be read and understood by people.

The first type of processor is called an *assembler*. It is a one-for-one translator; one coded instruction becomes one instruction to the computer. The assembler-level language on PDP-11 computers is called MACRO-11. It is slower to code and compile MACRO-11 programs than just about any other language, but in return the programmer gets considerably more control over the actual operation of a program than is possible under other languages.

Compilers and *interpreters* are the other two types of language processors. As opposed to the assembler-level language, these process what are always called the higher-level languages, the languages with such familiar names as FORTRAN, BASIC, and COBOL. In addition to such industrywide languages, Digital-specific languages such as DATATRIEVE-11 and DIBOL-83 are also higher-level languages.

A fundamental difference between compilers and assemblers is the number of machine instructions that may be represented by a single language instruction. It may be, for example, that a single FORTRAN command is compiled into 20 or more machine instructions. Of course, this speeds up the coding process immensely, but it also means that the programmer must relinquish some of the control over program execution and environment that would be available to the MACRO-11 program.

Most compilers do not translate the *source code* that the programmer has written until they read the program all the way through at least once. Several passes over the source code are used to produce what is called *object code*, the form of binary formats that the machine can actually execute. Such multipass compilation allows the compiler to eliminate unnecessary code called code optimization and to perform many levels of error checking. A virtue of error checking at compilation time is that far fewer errors are actually encountered in the execution of the program. Thus, programmer time is more efficiently spent and computer resources are better used.

Interpreters translate source statements immediately into a format that the machine can interpret. The option for code optimization is lost, but this is balanced by the ability to execute programs on a statement-by-statement basis. Program development is enhanced in an interpreter, since there is an immediate response from the computer to the programmer in the case of detected errors. An entire program need not be read to find one level of error. In many situations, this is preferable to waiting until the entire program is compiled.

Most operating systems offered by Digital can support a variety of language processors. It is unlikely that a particular installation would require all the compilers and interpreters available from Digital, but it might have several, such as FORTRAN, DATATRIEVE, and BASIC. Language processors are usually layered products, purchased in addition to the operating system. For example, COBOL is a layered product for RSX, but BASIC-PLUS is bundled with RSTS/E.

In many cases, application programs need not be written exclusively in a single language. For example, it may happen that a specific operation, such as the management of I/O devices (by I/O drivers) is best accomplished by programs written in assembler-level language, while the rest of the program is coded in COBOL. The driver may be coded in its most efficient format and later incorporated into the compiled COBOL object code. The complex details of this type of operation are handled by the operating system and the language processors, and are transparent to the programmer.

Many of the actual routines required by an application program are not written into the program. When the BASIC programmer asks the machine to extract a square root, for example, using the SQRT instruction in the program. Within the *Object Time System* (OTS) of the BASIC compiler is a mathematical functions library, which holds a square root algorithm. The SQRT instruction causes the algorithm to be called up into the program and to run on the appropriate variable. Since there are many cases in which it is simpler to include commonly used routines in programs than to rewrite them from scratch, each compiler is equipped with an Object Time System, filled with frequently required routines and functions. As the compilation process occurs, the locations of needed OTS routines are flagged. At the end of compilation, when the object code is ready, the appropriate routines from the OTS are inserted in the program at the flags. An interpreted language may also have an OTS, but it is more likely to be called at runtime, rather than at compiletime.

You may insert your own common routines into the OTS, so that your efficiency in coding is improved. Drivers for your specific devices, or algorithms for often-run procedures, can be programmed once, and then just called as necessary.

▪ Program Development

Program development refers to the operation of writing and checking workable computer programs. Obviously, there is more to it than merely writing the language code, but the more sophisticated the operating system, the easier will be the use of the facilities available for program development. In computer jargon, the more features the operating system provides to simplify the programmer's task, the "friendlier" the program development environment. PDP-11 computers provide a very friendly environment under most operating systems. The history of software improvement is often the history of making the full capabilities of the computer more and more readily available to users.

Some of the facilities of the program development environment are listed and described below. Not every operating system provides all such facilities. Table 1-2 provides a comparison across the operating systems provided for use with the PDP-11 family.

First among the program development utilities are the *editors*. An editor allows the addition, deletion, movement, and concatenation of text. It also provides capabilities for searching a text for specific character strings, for replacement of one string by another, and for most of the other text manipulation operations one might want to perform in any writing. In developing a program, the editors provide an easy way to create a file and to correct and alter programs, either for experimenting with new ideas or for changing programs as required by new application circumstances or by the discovery of errors.

The *debugger* is another utility of extreme usefulness to programmers. It vastly simplifies the task of checking a program for logical errors by letting the programmer "step through" the program and follow what is happening to the values of chosen variables at each step of the way. Both the debugger and the editor are usually used by the programmer right at a video or hardcopy terminal, in an interactive session.

The *linker* is the crucial utility that takes object files written (created) by the language processors and prepares them for execution. It does this in various ways for various machines, operating systems, and languages. Basically, the linker adjusts addresses of the modules that make up the program—both those modules in the source code and those drawn from the OTS and its libraries; the linker "resolves" addresses; that is, it arranges the modules in such a way that there is no inconsistency in the references among modules and within the segments of the program. The linker also organizes, defines, and resolves certain kinds of symbols used internally in the compilation and execution of computer programs.

The *librarian* is, just as the name implies, the utility that manages the creation, modification, and maintenance of libraries in the operating system.

Modularity is the term used to describe the division of a program into blocks of logically related material. Very large programs might be modularized in order to compile efficiently, or to run efficiently. Complex programs might be broken into modules for program development — code optimization, debugging. An advantage of modularized programs in this latter situation is that modules can be computed individually before linking, so that an error requires only the recompilation of one module rather than of the whole program.

Chapter 11 • MACRO

MACRO-11 — The Assembly-level Language for PDP-11s

PDP-11 MACRO (or MACRO-11) is a fast, compact assembly language that gives the programmer complete control over the environment in which a program is developed and executed. PDP-11 MACRO processes source programs written in the MACRO assembly language and produces a relocatable object module and optional assembly listing. MACRO-11 is included with the RSX-11M-PLUS, RSX-11M, RSTS/E, RT-11, and IAS operating systems, as well as VAX/VMS.

PDP-11 MACRO provides for

- Global symbols for linking separately assembled object programs. (This promotes modular program design.)
- Device and file name specifications for input and output files.
- User-defined macros.
- Comprehensive system macro library.
- Program sectioning directives.
- Conditional assembly directives.
- Assembly and listing control functions at program and command string levels.
- Alphabetized, formatted symbol table listing.
- Default error listing on command output device.

The MACRO assembler for all operating systems also features

- Global arithmetic, global assignment operator, global label operator, and default global declarations.
- Multiple macro libraries with fast access structure.
- Predefined (default) register definitions.

▪ MACRO Program Structure Processes Source Statements Sequentially

A MACRO source program is composed of a sequence of source coding lines, each of which contains a single assembly language statement followed by a statement terminator, such as a carriage return. The assembler processes source statements sequentially, generating binary machine instructions and data words or performing assembly-time operations (such as macro expansions) for each statement.

A statement can contain up to four fields, identified by order of appearance and by specified terminating characters. The general statement is

```
label: operator operand(s) ;comments
```

of which the label and comment fields are optional. Operator and operand fields are interdependent: either can be be omitted depending on the contents of the other.

A label is a unique user-defined symbol that is assigned the value of the current location counter and entered into the user-defined symbol table. It provides a symbolic means of referring to a specific location within a program. The value of the label can be either absolute (fixed in memory independent of the position of the program) or relocatable (not fixed in memory), depending on whether the location counter value is currently absolute or relocatable.

Comments do not affect assembly processing or program execution, but are useful in source listings for later analysis, documentation, or debugging.

An operator field can contain a macro call, a PDP-11 instruction mnemonic, or an assembler directive. When the operator is a macro call, the assembler inserts the appropriate code during assembly to expand the macro; for an instruction mnemonic, it specifies the instruction to be generated and the action to be performed on any operands which follow; and when the operator is an assembler directive, it specifies a certain function or action to be performed during assembly. Operands can be expressions, numbers, symbolic arguments, or macro arguments.

Some statements have no operands:

```
BPT
```

Some statements have one operand

```
CLR R0
```

while others have two

```
MOV #344,R2
```

Symbols and Symbol Definitions

Three types of symbols can be defined for use within MACRO source programs: permanent symbols, user-defined symbols, and macro symbols. Correspondingly, MACRO maintains three types of symbol tables: the Permanent Symbol Table (PST), the User Symbol Table (UST), and the Macro Symbol Table (MST).

Permanent symbols consist of the PDP-11 instruction mnemonics and assembler directives. Also, the assembler has REGISTER names predefined for R0 to R5, SP (stack pointer), and PC (program counter). The PST contains all the permanent symbols automatically recognized by MACRO; it is part of the assembler itself. Since these symbols are permanent, they do not have to be defined by the user in the source program.

User-defined symbols are those given as labels or defined by direct assignment, while macro symbols are those symbols used as macro names. The UST and MST are constructed during assembly by adding the symbols to the UST or MST as they are encountered. To determine the value of the symbol, the assembler searches the three symbol tables; for opcodes, the search order is MST, PST, UST; for operands, the search order is UST, PST.

The search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols, so that the same name can be assigned to both a macro and a label.

User-defined symbols are either *internal* or *external* (global) to a source program module. An internal symbol definition is limited to the module in which it appears. A global symbol can be defined in one source program module and referenced within another.

Internal symbols are temporary definitions, resolved by the assembler. Global symbols are preserved in the object module and are not resolved until the object modules are linked into an executable program. With some exceptions, all user-defined symbols are internal unless explicitly defined as global.

A direct assignment statement with the general format *symbol expression* associates a symbol with a value.

By using two equal signs instead of one, the symbol is declared a global symbol. *Expressions* are combinations of terms that are joined together by binary operators — +, −, *, ÷, & (logical AND), ! (logical OR)—and that reduce to a 16-bit value.

Local symbols are specially formatted internal symbols used as labels within a given range of source code, called a local symbol block. They have the form n$, where n is a decimal integer between 1 and 65, 535, inclusive, for example, 1$, 27$, 59$, 104$.

Local symbols provide a convenient means of generating labels to be referenced only within a local symbol block. Their use reduces the possibility of entry point symbols with multiple definitions. Because a local symbol may not be referenced from other source program modules or even from outside its local symbol block, local symbols of the same name can appear in other local symbol blocks without conflict.

Directives

A program statement can contain one of three different operators: a macro call, a PDP-11 instruction mnemonic, or an assembler directive. MACRO includes directives for

- Listing control.
- Function specification.
- Data storage.
- Radix and numeric usage declarations.
- Location counter control.
- Program termination.
- Program boundaries information.
- Program sectioning.
- Conditional assembly.
- Macro definition.
- Macro deletion.
- Macro attributes.
- Macro message control.
- Repeat block definition.
- Macro libraries.
- File control.
- Symbol control.

The sections that follow illustrate some of the capabilities of the various classes of directives.

Listing Control Directives

Several directives are provided to control the content, format, and pagination of all listing output generated during assembly. Facilities also exist for creating object module names and other identification information in the listing output.

The listing control options can also be specified at assembly time through options included in the listing file specification in the command string issued to the MACRO assembler. The use of these options overrides all corresponding listing control directives in the source program.

When no listing file is specified, any errors encountered in the source program are printed on the terminal from which MACRO was initiated.

Function Directives

Function control options are available through the .ENABL and .DSABL directives. These directives are included in a source program to invoke or inhibit certain MACRO functions and operations incidental to the assembly process. They include

- Produce absolute binary output.
- Assemble all relative addresses as absolute addresses. This function is useful during the debugging phase of program development.
- Cause source columns 73 and greater (to the end of the line) to be treated as comment. The most common use of this feature is to permit sequence numbers in card columns 73-80.
- Truncate or round floating-point literals.
- Accept lowercase ASCII input instead of converting it to uppercase.
- Enable a local symbol block to cross program section boundaries.
- Inhibit binary output.
- Inhibit the normal default register definitions.
- Treat all undefined symbol references as default global references.

Conditional Assembly Directives

Conditional assembly directives enable the programmer to include or exclude blocks of source code during the assembly process, based on the evaluation of stated condition tests within the body of the program. This allows a programmer to generate several variations of a program from the same source.

The programmer can define a conditional assembly block of code, and within that block, issue subconditional directives. Subconditional directives indicate

- The assembly of an alternate body of code when the condition of the block tests false.
- The assembly of a noncontiguous body of code within the conditional assembly block, depending on the result of the conditional test on entering the block.
- The unconditional assembly of a body of code within a conditional assembly block.

Conditional assembly directives can be nested to 16 levels.

Macro Definitions and Repeat Blocks

In assembly language programming, it is often convenient and desirable to generate a recurring coding sequence by invoking a single statement within the program. In order to do this, the desired coding sequence is first established with dummy arguments as a macro definition. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the macro definition) generates the desired coding sequence or macro expansion.

MACRO can automatically create unique local symbols. This automatic facility is invoked on each call of a macro whose definition contains a dummy argument preceded by the question mark (?) character if a real argument of the macro call is either null or missing.

An indefinite repeat block is a structure that is very similar to a macro definition. Such a structure is essentially a macro definition that has only one dummy argument. At each expansion of the indefinite repeat range, this dummy argument is replaced with successive elements of a specified real argument list. An indefinite repeat block directive and its associated repeat range are coded inline within the source program. This type of macro definition does not require calling the macro by name.

An indefinite repeat block can appear within or outside another macro definition, indefinite repeat block, or repeat block.

Macro Calls and Structured Macro Libraries

All macro definitions must occur prior to their references within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required in the program. (System macros include the monitor programmed requests or executive directives available with each operating system.)

The .MCALL directive is used to specify the names of all the macro definitions not defined in the current program but used in the program. When this directive is encountered, MACRO searches the system macro library file to find the requested definition.

MACRO extends this macro call facility by enabling the programmer to retrieve macros from libraries of user-defined macros. The .MCALL directive provides the means to access both user-defined and system macro libraries during assembly.

The MACRO assembler assumes that the system macro library and user-defined macro libraries are constructed in a special direct-access format to retrieve macro definitions quickly. These structured macro libraries are created by the Librarian utility program.

Each library file contains an index of the macro definitions it contains. When an .MCALL directive is encountered in the source program, MACRO searches the user macro libraries for the named macro definitions, and, if necessary, continues the search with the system macro library. Because each macro library contains an index of all of its entries, MACRO searches only the index in each library to find where the macro definition is stored.

Macro libraries to be searched may be specified by both the initial MACRO-11 command line and by use of the .LIBRARY directive. The .MDELETE directive may be used to delete a macro once used. This conserves dynamic memory.

Program Sectioning Directives

The .PSECT directive is used to declare names for program sections and to establish certain program section attributes. These program section attributes are used when the program is linked into an executable load module or task.

A program can consist of an absolute program section, an unnamed relocatable program section, and up to 254 named relocatable or absolute program sections. Absolute program sections link the program with fixed memory locations, such as interrupt vectors and the peripheral device register addresses, as well as define values of constants.

The relocatable program sections are not fixed at an absolute address. Instead, symbols within a relocatable section are defined relative to the start of that section. The programmer specifies the overall ordering of relocatable .PSECTS, but the task builder (or linker) resolves the final addresses of the .PSECTS according to their attributes.

By maintaining separate location counters for each program section, MACRO allows the user to create data structures that are not physically contiguous within the program, but which can be linked contiguously following assembly.

The programmer can save the current .PSECT context with a .SAVE directive, and later restore that context with a .RESTORE directive.

The .PSECT directive allows the user to exercise absolute control over the memory allocation of a program at task-build time, since any program attributes established through this directive are passed to the Task Builder. For example, if a programmer is writing programs for a multiuser environment, a program section containing pure code (instructions only) or a program section containing impure code (data only) can be explicitly declared through the .PSECT directive. Furthermore, these program sections can be explicitly declared as read-only code, qualifying them for use as protected, reentrant programs. In addition, program sections exhibiting the global attribute can be explicitly allocated in a task's overlay structure by the user at task build time. The advantages gained through sectioning programs in this manner relate primarily to control of memory allocation, program modularity, and more effective partitioning of memory.

The .PSECT directive allows the user to define the following program section attributes:

- *Access*—Two types of access can be permitted to the program section: read-only or read/write. RSX-11M-PLUS and IAS support read-only access by setting hardware protection for the program section.
- *Contents*—A program section can contain either instructions or data. This attribute allows the Task Builder to differentiate global symbols that are program entry-point instructions from those that are data values.
- *Scope*—The scope of the program section can be global or local. In building single-segment programs, the scope of the program has no meaning because the total memory allocation for the program will go into the root segment of the task. The global or local attribute is significant only in the case of overlays. If an object module contains a local program section, then the storage allocation for that module will occur within the segment in which the module resides. Many modules can reference this same program section, and the memory allocation for each module is either concatenated or overlaid within the segment, depending on the argument of the program section defining its allocation requirements (see below). If an object module contains a global program section, the memory area allocations to this program section are collected across segment boundaries, and the allocation of memory for that section will go into the segment nearest the root in which the first memory allocation to this program section appeared.
- *Relocatability*—A program section can be absolute or relocatable. When a program section is declared to be absolute, the program section requires no relocation. The program section is assembled and loaded, starting at absolute virtual address 0. When the program section is declared to be relocatable, the Task Builder calculates a relocation bias and adds to it all references within the program section.

- *Allocation Requirements* — The program section can be concatenated or overlaid. When concatenated, all memory allocations to the program section are to be concatenated with other references to this same program section in order to determine the total memory allocation requirements for this program section. When overlaid, all memory allocations to the program section are to be overlaid. Thus, the total allocation requirement for the program section is equal to the largest individual allocation request for this program section.

▪ MACRO Accepts Source Data from Any Input Device

The MACRO assembler can accept source data from any input device. The sources to be included in a single assembly are listed in the command string from left to right in the order of assembly. The last statement in the last source specified is normally the .END statement, but if the .END statement is not provided, it is assumed.

MACRO is a two-pass assembler. During pass one, MACRO locates and reads all required macros from libraries, builds symbol tables and program section tables for the program, and performs a rudimentary assembly of each source statement. During pass two, MACRO completes the assembly, writes out an object file, and generates an assembly and symbol table listing for the program.

The object module MACRO produces must be processed by the operating system's linker utility program (called Linker or Task Builder) to create an executable program. The linker joins separately assembled object modules into a single load module (or task image). The linker fixes (makes absolute) the values of the external or relocatable symbols in the object module.

To enable the linker to fix the value of an expression, MACRO passes it certain directives and parameters. In the case of the relocatable expressions in the object module, the linker adds the base of the associated relocatable program section to the value of the relocatable expression provided by MACRO. In the case of external expression values, the linker determines the value of the external term in the expression (since the external expression must be defined in at least one of the other object modules being linked together) and then adds it to the absolute portion of the external expression, as provided by MACRO.

In summary, an executable program image can be constructed from one or more source modules, which can be assembled either separately or together. The resultant object module(s) must be linked together using the linker utility. Figure 11-1 illustrates the processing steps required to produce an executable program from several sources stored as files.

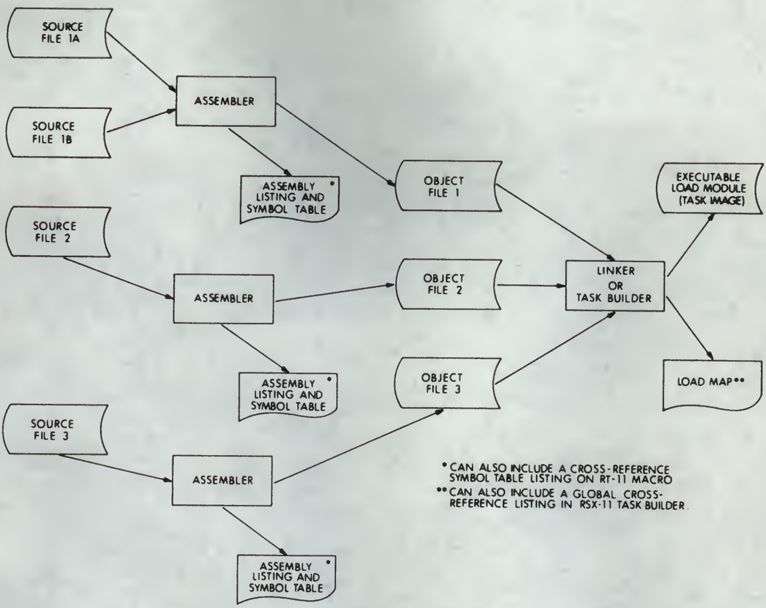


Figure 11-1 ▪ MACRO Assembly Procedure

Chapter 12 • FORTRAN

▪ FORTRAN Is Ideal for Manipulating and Performing Calculations of Numeric Data

FORTRAN was developed in the mid-1950s specifically to handle scientific applications involving large amounts of computation. Since then, FORTRAN has evolved into one of the most widely used languages, with applications in realtime control (scientific experiments, industrial processes, data collection and reduction), computation (structural analysis, simulation and modeling, electronic design, heavy computing data reduction), and general data processing (maintenance of databases and report generation). Because of its traditional preeminence in certain markets and its long, stable history, FORTRAN continues to be studied by most people specializing in computer and information science in college.

Digital offers two versions of FORTRAN for use on its PDP-11 computers. The first, PDP-11 FORTRAN-77, conforms to the most recent ANSI FORTRAN standard, X3.9-1978 (commonly referred to as FORTRAN-77), at the subset language level. Earlier versions of PDP-11 FORTRAN-77 were called PDP-11 FORTRAN IV-PLUS and were based on the 1966 ANSI standard. PDP-11 FORTRAN-77 runs under the RSX-11M-PLUS, RSX-11M, MicroRSX, RSTS/E, MicroRSTS, IAS, and P/OS operating systems, as well as under VAX/VMS when using VAX-11 RSX. It produces machine code highly optimized for execution on a PDP-11 with a floating-point processor. PDP-11 FORTRAN-77 features optimization techniques that improve memory efficiency and increase program execution speed.

The second FORTRAN offering, FORTRAN IV, is based on an earlier ANSI FORTRAN standard (X3.9-1966). This FORTRAN language works on RSX-11M-PLUS, RSX-11M, RT-11, RSTS/E, and IAS operating systems as well as on VAX/VMS under VAX-11 RSX. FORTRAN IV is characterized by high compilation speed and efficiency in small memory environments.

▪ The FORTRAN-77 Compiler Produces Optimized Machine Code

The PDP-11 FORTRAN-77 compiler accepts a source program and produces a relocatable object module and optionally a listing file as output. PDP-11 FORTRAN-77 is designed to minimize the size and increase the speed of executable programs. It accomplishes this through extensive optimizations such as subexpression elimination, peephole optimizations, removal of invariant expressions from DO loops, and allocation of processor registers across block IF constructs and DO loops. The FORTRAN-77 compiler provides optional, switch-selectable support for programs conforming to the previous ANSI FORTRAN standard X3.9-1966. Programs that successfully compile using the PDP-11 FORTRAN-77 compiler can be compiled using VAX FORTRAN without modification to the source code. Programs that successfully compile using PDP-11 FORTRAN-IV can be compiled using either FORTRAN-77 or VAX FORTRAN by setting the NOF77 switch.

▪ FORTRAN-77 Conforms to the Subset Language and Includes Features from the Full Language

PDP-11 FORTRAN-77 includes the following features of full-language FORTRAN as defined by the ANSI FORTRAN standard X3.9-1978:

- Double-precision and complex data types.
- Function subprograms, including LEN, ICHAR, and INDEX.
- Exponentiation forms, including double precision.
- Format edit descriptors, including S, SP, SS, T, TL, and TR.
- Generic function selection based on argument data type for FORTRAN-defined functions.
- Use of any arithmetic expression as the initial value, increment, or final value in a DO statement.
- Use of a real or double-precision variable as a DO statement control variable.
- CLOSE and OPEN statements.
- Use of the specification ERRs in READ or WRITE statements to transfer control when an error occurs to the statement specified by S.
- Use of list-directed I/O to perform formatted I/O without a format specification.
- Use of constants and expressions in the I/O lists of WRITE, REWRITE, TYPE, and PRINT statements.

- Specification of lower bounds for array dimensions in array declarators.
- Use of ENTRY statements in SUBROUTINE and FUNCTION subprograms to define multiple entry points.
- Use of PARAMETER statements to assign symbolic names to constant values.

Language Extensions Go beyond the Standard

The following language extensions beyond the ANSI FORTRAN standard X3.9-1978 are included in PDP-11 FORTRAN-77:

- You can use any arithmetic expression as an array subscript. If the expression is not an integer type, it is converted to integer type.
- Mixed-mode expressions can contain elements of any data type except character.
- The LOGICAL*1 and LOGICAL*2 data types have been added.
- The IMPLICIT statement redefines the implied data type of symbolic names.
- The following input/output statements have been added:

ACCEPT TYPE PRINT	Device-oriented I/O
READ (u'r) WRITE (u'r) FIND (u'r)	Unformatted direct-access I/O
READ (u'r,fmt) WRITE (u'r,fmt)	Formatted direct-access I/O
DEFINE FILE	File control and attribute specification
ENCODE DECODE	Formatted data conversion in memory
READ (u,f,key) READ (u,key)	Indexed I/O
REWRITE DELETE UNLOCK	Record control and update

- You can include any explanatory comment on the same line as any statement. These comments begin with an exclamation point (!).
- You can include debugging statements in a program by placing the letter D in column 1. These statements are compiled only when you specify a compiler command qualifier; otherwise, they are treated as comments.
- You can use any arithmetic expression as the control parameter in the computed GOTO statement.

- VIRTUAL arrays, on all PDP-11 operating systems except RSTS/E, provide large data areas outside of normal program address space.
- You can include the specification ERR=s in any OPEN, CLOSE, FIND, DELETE, UNLOCK, BACKSPACE, REWIND, or ENDFILE statement to transfer control to the statement specified by the letter s when an error condition occurs.
- The INCLUDE statement incorporates FORTRAN statements from a separate file into a FORTRAN program during compilation.
- ENCODE and DECODE statements transfer data between variables or arrays in internal storage, and translate that data from internal to character form, or from character to internal form, according to format specifiers.
- The INTEGER*4 data type provides a sign bit and 31 data bits.
- You can use hexadecimal and octal constants in place of any numeric constants.
- O and Z format edit descriptors.
- You can use character substrings and all the character intrinsic functions defined in the full language except CHAR.

FORTRAN Programs Consist of Statements and Optional Comments

Statements are organized into program units. Program units are sequences of statements that define a computing procedure and terminate with an END statement. A program unit can be a main program or a subprogram. An executable program consists of one main program and one or more optional subprograms.

Statements fall into two general classes—executable and nonexecutable. Executable statements specify the actions of a program; nonexecutable statements describe data arrangement and characteristics, and provide editing and data-conversion information as well.

Statements are also divided into physical sections called lines. A line is a string of as many as 80 characters. If a statement is too long to fit on one line, it can be continued on additional lines, called continuation lines.

A label identifies a statement so other statements can transfer control to it or get the information it contains. The label is an integer placed in the first five columns of a statement's initial line. Any statement can have a label; however, only executable and FORMAT statements can be referenced with a label.

Comments don't affect program processing in any way; they are merely a documentation aid. Comments can describe the action of a program, identify program sections and processes, and help in reading the source program listings. Any printable character can appear in a comment.

▪ The Object Time System Helps Build Tasks Ready for Execution

The FORTRAN-77's Object Time System (OTS) is a library of routines that are selectively linked with compiler-produced object modules by the operating system's taskbuilder to produce a task ready for execution. The PDP-11 FORTRAN-77 OTS contains routines for I/O processing, task control, error processing, mathematical computation, and system subroutine access. By selective linking, if a program performs only sequential formatted I/O, none of the direct-access I/O routines are included in the task.

The OTS is composed of the following routines:

- Math routines, including the FORTRAN-77 library functions and other arithmetic routines (e.g., exponentiation routines).
- Miscellaneous utility routines (e.g., ASSIGN, DATE, ERRSET).
- Routines that handle FORTRAN-77 input/output.
- Error-handling routines that process arithmetic errors, I/O errors, and system errors.
- Miscellaneous routines required by the compiled code.

PDP-11 FORTRAN-77 is distributed with both of the following object time systems.

- The OTS based on File Control Services (FCS) is a package of routines that can handle many file operations transparently to the user, and allows sequential and random access to sequentially organized files.
- The OTS based on Record Management Services (RMS) uses RMS to provide access to sequential, relative, and indexed files.

▪ FORTRAN-77 Optimizations Increase Execution Efficiency

Optimizations are techniques used to increase the execution efficiency of an object program. PDP-11 FORTRAN-77 optimizations include

- Constant pooling—Storage is allocated for only one copy of a constant in the compiled program. Constants, including most numeric constants, used as immediate-mode operands are not allocated storage.
- Peephole optimizations—The initial machine instructions generated by a FORTRAN program are examined to find operations that can be replaced by shorter, faster code sequences. The final code generated by the compiler contains these improved code sequences.
- Common subexpression elimination—Often the same subexpression appears in more than one computation. If the values of the operands of a common subexpression are not changed between computations, they can be computed once and substituted wherever the subexpression appears.
- Removal of invariant expressions from DO loops—An algorithm executes faster if computations are moved from frequently executed program sequences to less frequently executed program sequences. In particular, computations within a loop involving only constants can be moved outside the loop.
- Allocation of processor registers across block IF constructs and DO loops—Wherever possible, frequently referenced variables are retained in registers to reduce the number of load and store instructions executed. Frequently used variables and expressions are also assigned to registers across block IF constructs and DO loops.
- Shareable code—For the RSX-11M-PLUS operating system, the compiler produces shared object code as a compiletime option. Shared tasks may then be created by using the multiuser taskbuilder switch. This improves memory utilization in multiuser systems because many users share one memory-resident task.

▪ FORTRAN Offers a Choice of Debugging Tools

FORTRAN-77 DEBUG is a symbolic debugger available with FORTRAN-77. In addition, two standard debugging facilities are available to FORTRAN-77—the FORTRAN Object Time System and the other that allows a FORTRAN statement to be conditionally compiled.

The *FORTRAN Object Time System* provides a traceback feature for fatal runtime errors. This feature locates the actual program unit and line number of a runtime error. Immediately following the error message, the error handler will list the line number and program unit name in which the error occurred. If the program unit is a subroutine or function subprogram, the error handler will trace back to the calling program unit and display the name of that program unit and the line number where the call occurred. This process will continue until the calling sequence has been traced back to a specific line number in the main program. This allows the exact determination of the location of an error even if the error occurs in a deeply nested subroutine.

In addition to the FORTRAN OTS error diagnostics that include the traceback feature, another debugging tool is available. The letter D in column 1 of a FORTRAN statement allows that statement to be conditionally compiled. These statements are considered comment lines by the compiler unless the appropriate debugging lines switch is issued in the compiler command string. In this case, the lines are compiled as regular FORTRAN statements. Liberal use of the PAUSE statement and selective variable printing can provide the programmer with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program testing stages and later eliminated without source program modification.

▪ FORTRAN-77 DEBUG Helps You Find Errors

FORTRAN-77 DEBUG is a symbolic debugger designed to help find logic and programming errors in a successfully compiled FORTRAN-77 and MACRO-11 program that doesn't run correctly. The program may be terminating abnormally, giving incorrect output, or going into an infinite loop.

Abnormal termination of a program suggests either a logical error or an error caused by something in the environment. Because abnormal termination usually occurs reasonably soon after the error that caused it, a useful approach to locating the error is to examine the output of the program at points just before termination. This technique does not always work, however, when the error causes infinite looping or when a pointer error leads the program to an incorrect location. In these cases, forward tracing is the best method. The debugger can advance the program in predetermined steps, examining locations as it goes. When the debugger encounters the unexpected output, it can be used to isolate the area of code that caused it.

FORTRAN-77 DEBUG is a powerful and flexible tool that helps find errors in a program. It has the following features:

- It is interactive. You issue debugger commands from your terminal and see their effects immediately.
- It is symbolic. You can refer to program locations by the symbols you used for them in your program. The debugger output uses symbols wherever possible.
- It understands variable names and data types and will accept and display data in a variety of formats.
- It gives online help. When you type HELP, the debugger responds with a list of commands and related features to help you debug.

FORTRAN-77 DEBUG commands include

- file-spec
- CANCEL
- DEFINE
- DEPOSIT
- DISABLE
- ENABLE
- EVALUATE
- EXAMINE
- EXIT
- GO
- HELP
- SET
- SHOW
- STEP
- UNDEFINE

FORTRAN IV Optimizes Code and Simplifies Programming

PDP-11 FORTRAN IV is based on the previous specification for ANSI FORTRAN, X3.9-1966. The following are enhancements in FORTRAN IV not found in this standard:

- Array subscripts — Any arithmetic expression can be used as an array subscript. If the value of the expression is not an integer, it is converted to integer type.
- Array dimensions — Arrays can have up to seven dimensions.
- Alphanumeric literals — Strings of characters bounded by apostrophes can be used in place of Hollerith constants.
- Mixed-mode expressions — Mixed-mode expressions can contain any data type, including complex and byte.
- End-of-line comments — Any FORTRAN statement can be followed, in the same line, by a comment beginning with an exclamation point.
- Debugging statements — Statements that are included in a program for debugging purposes can be so designated by the letter D in column 1. Those statements are compiled only when the associated compiler command string option switch is set. They are treated as comments otherwise.
- Read/Write end-of-file or error condition transfer — the specifications end = n and err = n (where n is a statement number) can be included in any read or write statement to transfer control to the specified statement upon detection of an end-of-file or error condition. The err = n option is also permitted in the encode and decode statements, allowing program control of data format errors.
- General expressions in I/O lists — general expressions are permitted in I/O lists of write, type, and print statements.
- General expression DO and GOTO parameters — General expressions are permitted for the initial value, increment, and limit parameters in the DO statement, and as the control parameter in the computed GOTO statement.
- DO increment parameter — The value of the DO statement increment parameter can be negative.
- Optional statement label list — The statement label list in an assigned GOTO is optional.

- Override field width specifications—Undersized input data fields can contain external field separators to override the FORMAT field-width specifications for those fields (called "short field termination"), permitting free-format input from terminals.

- Default FORMAT widths—Specifying input or output formatting by type and default-width supplies precision values to the programmer.

- Additional I/O statements(u = logical unit number; r = record number):
  - File control and attribute definition
    OPEN
    CLOSE
  - List-directed (free format)
    READ (u,*)
    WRITE (u,*)
    TYPE*
    ACCEPT*
    PRINT*
  - Device-oriented I/O
    ACCEPT
    TYPE
    PRINT
  - Memory-to-memory formatting
    ENCODE
    DECODE
  - Unformatted direct access I/O
    DEFINE FILE
    READ (u'r)
    WRITE (u'r)
    FIND (u'r)

The unformatted direct access I/O facility allows the FORTRAN programmer to read and write files written in any format.

- Logical operations on INTEGER data—The logical operators .AND., .OR., .NOT., .XOR., and .EQV. may be applied to integer data to perform bit masking and manipulation.

- Additional data type—The byte data type (keyword LOGICAL*1 or BYTE) is useful for storing small integer values as well as for storing and manipulating character information.

- IMPLICIT declaration—IMPLICIT redefines the implied data type of symbolic names.

▪ A Fast Compiler Allows Tradeoffs

The PDP-11 FORTRAN IV compiler and Object Time System are available as an optional language processing system on the RSX-11M-PLUS, RSX-11M, RSTS/E, RT-11, and IAS operating systems, as well as on VAX/VMS under VAX-11 RSX. The compiler accepts source programs written in the FORTRAN IV language and produces an object file that must be linked prior to execution.

A fast, one-pass compiler, the FORTRAN IV compiler has options that allow program size (threaded code) versus execution speed (inline code) tradeoffs. FORTRAN IV compiler optimizations include

- Common subexpression elimination
- Local code tailoring
- Array vectoring
- Optional inline code generation for integer and logical operations

Despite its small size requirements and high compilation rate, FORTRAN IV provides a high level of automatic object program optimization. The compiler performs redundant expression elimination, constant expression folding, branch structure optimization, and several types of subscripting optimizations.

FORTRAN IV has no statement-ordering requirements; therefore, declarations can appear anywhere within the source program. Terminal format input (using the tab character to delimit field) makes program preparation easier.

In order to allow larger FORTRAN programs, FORTRAN IV can allocate array storage outside a program's logical address space. Such arrays are called virtual arrays and can contain any data type, but they may also require operating system support of memory management directives. Virtual arrays can be used with all PDP-11 operating systems except RSTS/E.

▪ An Object Time System Contains Common Sequences of Machine Instructions

A few executable FORTRAN statements can be translated directly into machine instructions. Typical FORTRAN operations, however, require long sequences of PDP-11 machine instructions. Standard sequences, for example, are needed to locate an element of a multidimensional array, initialize an I/O operation, or simulate a floating-point operation not supported by the hardware configuration.

The common sequences of PDP-11 machine instructions are contained in the Object Time System (OTS) library. The FORTRAN IV compiler does not always generate pure machine instructions for the FORTRAN source code statements. When using the threaded code option, it simply determines which combination of appropriate OTS routines is needed to implement a FORTRAN program. During the linking process for an object program, the linker utility incorporates the needed OTS routines into the load module. During program execution, these routines are chained together to effect the desired result. However, inline code is used for improved execution speed for some operations where appropriate.

During compilation, FORTRAN IV performs ten categories of program optimization.

Briefly, they are

- Compiled FORMAT statements — FORMAT statements are translated into internal form at compiletime, increasing execution speed and decreasing program size.
- Array vectoring — Provides for faster location of array elements in multidimensional arrays.
- Constant folding — Integer constant expressions are evaluated at compiletime.
- Constant subscript evaluation — Constant subscript expressions in array calculations are evaluated at compiletime.
- Unreachable code elimination — Unreachable statements are eliminated from the object code.
- Common subexpression elimination — Redundant subexpressions whose operands do not change between computations are replaced by temporary values calculated only once.
- Peephole optimizations — Sequences of operations are replaced with shorter and faster equivalent operations.
- Branch optimization for arithmetic and Logical IF — Branch structures can be speeded up and decreased in size.
- Register allocation — Register allocation is improved to minimize direct memory references for variables.
- Loop optimization — Expressions dependent on loop index variables are replaced with less complex arithmetic operations.

▪ The Compiler Works Fast without Temporary Files

Instead of using temporary files to process source programs, the FORTRAN IV compiler performs all its activities in main memory. It reads the entire source program once, stores it in memory in a compacted format, and processes the compacted code in memory. Because a disk device is not used for temporary file operations, compilation speed is significantly increased.

To reduce the memory requirements of such a compilation system, the FORTRAN IV compiler employs a multiphase overlaid structure. The compiler consists of a large number of overlays. Most of the space allocated to the compiler is occupied by the compressed source code.

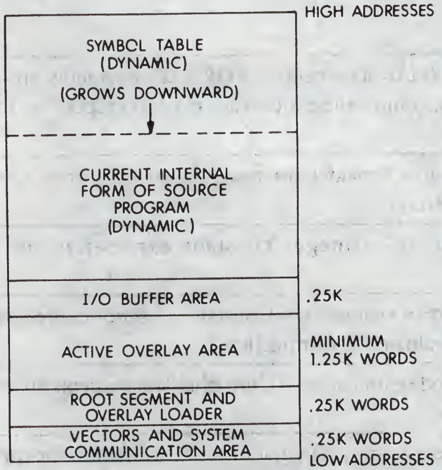


Figure 12-1 ▪ Compiletime Memory Map

The compiler begins by reading in as much of the source program as can fit in memory. It then compresses the source code in memory by removing blanks and other unnecessary data. It continues to read in more source code, compressing it as it goes, until the entire program segment fits in memory.

Once the source code is compressed into memory, the compiler begins processing the internal form of the source code as a whole. Because the entire program segment is available to the compiler, FORTRAN IV does not require statement ordering restrictions.

▪ FORTRAN IV Works across a Spectrum of Operating Systems

Though the compiler operation and facilities under all operating systems are essentially identical, each operating system provides additional features peculiar to the environment. The monitor programmed requests or executive directives, for example, are usually available as a library of FORTRAN-callable routines.

Under RT-11

The entire PDP-11 FORTRAN IV language processing system is operational in 16 Kbytes under the RT-11 SJ, FB, or XM monitors. To support strings, 32 Kbytes of memory are required.

The RT-11 System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines that allow a FORTRAN user to utilize various features of the RT-11 Foreground/Background (;dsus;) and Single-Job (SJ) monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and two-word integer support. SYSLIB is provided as a library of object modules to be combined with FORTRAN programs at link-time. SYSLIB allows the RT-11 FORTRAN user to write almost all application programs in FORTRAN with no assembly language coding.

Also available under RT-11 are

- A library of FORTRAN-callable graphics routines supporting the VT11, GT40, GT42, and GT44 graphics hardware systems.
- Plotting support for the LV11 electrostatic printer/plotter
- Laboratory data acquisition and manipulation routines used in conjunction with the LPS-11 and AR11 laboratory peripheral hardware.
- The Scientific Subroutine Library, providing FORTRAN-language routines for mathematical and statistical applications.
- Stand-alone program execution (SIMRT).

Under RSTS/E

PDP-11 FORTRAN IV operates in interactive or batch mode under the RSTS/E monitor and RT-11 Runtime System. The FORTRAN IV language processing system includes the FORTRAN IV compiler, the Object Time System (OTS), and several utility programs.

The entire system (including compiler and optimization components) is completely functional in a 16-Kbyte user area. A system interface occupying 8 Kbytes of memory is shareable among all FORTRAN IV users on the system. In addition, the FORTRAN IV system provides overlay support for programs and data, allowing extremely large programs to be run in a small region of memory.

RSTS/E FORTRAN IV provides assembly language subprogram support, using the MACRO assembler. Although the assembly language subprogram cannot issue any monitor calls, MACRO provides the experienced user with a tool to further enhance computational performance.

Under RSX-11

In RSX-11M and RSX-11M-PLUS, the FORTRAN IV compiler runs in a minimum partition of 14 Kbytes. If run in a larger partition, it uses the extra space for program and symbol table storage.

An RSX-11 library consists of object modules. Two types of libraries exist — shared and relocatable.

Relocatable libraries are stored in files. Object modules from relocatable libraries are built into the task image of each task referencing the module. The Task Builder is used to include modules from relocatable libraries in a task image. When a library specification is encountered in the command string, those modules in the library that contain definitions of any currently undefined global symbols are included in the task image. The user can construct relocatable libraries of assembly language and FORTRAN routines using the Librarian utility.

Shared libraries are located in main memory and a single copy of each library is used by all referencing tasks. Access to a shared library is gained by specifying the name of the library in an option at taskbuild time. Shared libraries are built using the Task Builder. They must contain shareable (reentrant) code.

Each RSX-11 system has a system relocatable library. The system relocatable library is automatically searched by the Task Builder if any undefined global references are left after processing all user-specified input files. The FORTRAN OTS may be included in the system library and hence is loaded automatically with FORTRAN programs.

The RSX-11 system library provides FORTRAN-callable forms of most executive directives. The FORTRAN programmer can schedule the execution of tasks, communicate with concurrently executing tasks, and manipulate system resources through these calls.

Industrial Society of America (ISA) extensions for process I/O control are available in FORTRAN-callable format under RSX-11M. Support for laboratory and process control peripherals is also included.

Libraries Contain Functions and Routines

Both the FORTRAN-77 programmer and the FORTRAN IV programmer can create a library of commonly used assembly language and FORTRAN functions and subroutines. The operating system's librarian utility provides a library creation and modification capability. Library files can be included in the command string to the linker utility. The linker recognizes the file as a library file and links only those routines in the library that are required in the executable program. By default, the linker also automatically searches the FORTRAN system library for any other required routines.

FORTRAN Library Functions

ABS(X)	Real absolute value
IABS(X)	Integer absolute value
DABS(X)	Double-precision absolute value
CABS(Z)	Complex to Real, absolute value
FLOAT(I)	Integer to Real conversion
IFIX(X)	Real to Integer conversion
SNGL(X)	Double to Real conversion
DBLE(X)	Real to Double conversion
REAL(Z)	Complex to Real conversion
AIMAG(Z)	Complex to Real conversion
CMPLX(X,Y)	Real to Complex conversion
AINT(X)	Real to Real truncation
INT(X)	Real to Integer conversion
IDINT(X)	Double to Integer conversion
AMOD(X,Y)	Real remainder
MOD(I,J)	Integer remainder
DMOD(I,J)	Double-precision remainder
AMAX0(I,J,...)	Real maximum from Integer list
AMAX1(X,Y,...)	Real maximum from Real list
MAX0(I,J,...)	Integer maximum from Integer list
MAX1(X,Y,...)	Integer maximum from Real list
DMAX1(X,Y,...)	Double maximum from Double list
AMIN0(I,J,...)	Real minimum of Integer list
AMIN1(X,Y,...)	Real minimum of Real list
MIN0(I,J,...)	Integer minimum of Integer list
MIN1(X,Y,...)	Integer minimum of Real list
DMIN1(X,Y,...)	Double minimum from Double list
SIGN(X,Y)	Real transfer of sign
ISIGN(I,J)	Integer transfer of sign
DSIGN(X,Y)	Double-precision transfer of sign

DIM(X,Y)	Real positive difference
IDIM(I,J)	Integer positive difference
EXP(X)	e raised to the X power (X is Real)
DEXP(X)	e raised to the X power (X is Double)
CEXP(Z)	e raised to the Z power (Z is Complex)
ALOG(X)	Returns the natural log of X (X is Real)
ALOG10(X)	Returns the log base 10 of X (X is Real)
DLOG(X)	Returns the natural log of X (X is Double)
DLOG10(X)	Returns the log base 10 of X (X is Double)
CLOG(Z)	Returns the natural log of Z (Z is Complex)
SQRT(X)	Square root of Real argument
DSQRT(X)	Square root of Double-precision argument
CSQRT(Z)	Square root of Complex argument
SIN(X)	Real sine
DSIN(X)	Double-precision sine
CSIN(Z)	Complex sine
COS(X)	Real cosine
DCOS(X)	Double-precision cosine
CCOS(Z)	Complex cosine
TANH(X)	Hyperbolic tangent
ATAN(X)	Real arc tangent
DATAN(X)	Double-precision arc tangent
ATAN2(X,Y)	Real arc tangent of (X/Y)
DATAN2(X,Y)	Double-precision arc tangent of (X/Y)
CONJG(Z)	Complex conjugate
RAN(I,J)	Returns a random number between 0 and 1

▪ FORTRAN Library Functions

The table below lists the PDP-11 FORTRAN-77 generic functions and intrinsic functions (listed in the column headed "Specific Name"). Superscripts in the table refer to notes that follow the table.

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Square root[1] a(1/2)	1	SQRT	SQRT	Real	Real
			DSQRT	Double	Double
			CSQRT	Complex	Complex
Natural logarithm[2] Log(e)a	1	LOG	ALOG	Real	Real
			DLOG	Double	Double
			CLOG	Complex	Complex
Common logarithm[2] Log(10)a	1	LOG10	ALOG10	Real	Real
			DLOG10	Double	Double
Exponential e(a)	1	EXP	EXP	Real	Real
			DEXP	Double	Double
			CEXP	Complex	Complex
Sine[3] Sin a	1	SIN	SIN	Real	Real
			DSIN	Double	Double
			CSIN	Complex	Complex
Cosine[3] Cos a	1	COS	COS	Real	Real
			DCOS	Double	Double
			CCOS	Complex	Complex
Tangent[3] Tan a	1	TAN	TAN	Real	Real
			DTAN	Double	Double

(cont.)

1. The argument of SQRT and DSQRT must be greater than or equal to 0. The result of CSQRT is the principal value with the real part greater than or equal to 0. When the real part is 0, the result is the principal value with the imaginary part greater than or equal to 0.
2. The argument of ALOG, DLOG, ALOG10, and DLOG10 must be greater than 0. The argument of CLOG must not be (0.,0.).
3. The argument of SIN, DSIN, COS, DCOS, TAN and DTAN must be in radians. The argument is treated modulo 2*pi.

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Arc sine[4,5]	1	ASIN	Real	Real	
Arc sin a		ASIN	Double	Double	
		DASIN			
Arc cosine[4,5]	1	ACOS	ACOS	Real	Real
Arc cos a			DACOS	Double	Double
Arc tangent(5)	1	ATAN	ATAN	Real	Real
Arc tan a			DATAN	Double	Double
Arc tangent[5,6]	2	ATAN2	ATAN2	Real	Real
Arc tan a(1)/a(2)			DATAN2	Double	Double
Hyperbolic sine	1	SINH	SINH	Real	Real
Sinh a			DSINH	Double	Double
Hyperbolic cosine	1	COSH	COSH	Real	Real
Cosh a			DCOSH	Double	Double
Hyperbolic tangent	1	TANH	TANH	Real	Real
Tanh a			DTANH	Double	Double
Absolute value[7]	1	ABS	ABS	Real	Real
[a]			DABS	Double	Double
			CABS	Complex	Real
			IIABS	Integer*2	Integer*2
			JIABS	Integer*4	Integer*4
		IABS	IIABS	Integer*2	Integer*2
			JIABS	Integer*4	Integer*4

(cont.)

4. The absolute value of the argument of ASIN, DASIN, ACOS, and DACOS must be less than or equal to 1.
5. The result of ASIN, DASIN, ACOS, DACOS, ATAN, DATAN, ATAN2, and DATAN2 is in radians.
6. The result of ATAN2 and DATAN2 is 0 or positive when a(2) is less than or equal to 0. The result is undefined if both arguments are 0.
7. The absolute value of a complex number, (X,Y), is the real value:
   (X(2) + Y(2))(1/2)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Truncation[8] [a]	1	INT	IINT	Real	Integer*2
			JINT	Real	Integer*4
			IIDINT	Double	Integer*2
			JIDINT	Double	Integer*4
		IDINT	IIDINT	Double	Integer*2
			JIDINT	Double	Integer*4
		AINT	AINT	Real	Real
			DINT	Double	Double
Nearest integer[8] [a + .5*sign(a)]	1	NINT	ININT	Real	Integer*2
			JNINT	Real	Integer*4
			IIDNNT	Double	Integer*2
			JIDNNT	Double	Integer*4
		IDNINT	IIDNNT	Double	Integer*2
			JIDNNT	Double	Integer*4
		ANINT	ANINT	Real	Real
			DNINT	Double	Double
Fix (real-to-integer conversion)	1	IFIX	IIFIX	Real	Integer*2
			JIFXI	Real	Integer*4
Float (integer-to-real conversion)	1	FLOAT	FLOATI	Integer*2	Real
			FLOATJ	Integer*4	Real
Double-Precision float (integer-to-double conversion)	1	DFLOAT	DFLOTI	Integer*2	Double
			DFLOTJ	Integer*4	Double
Conversion to single-precision	1	SNGL	—	Real	Real
			SNGL	Double	Real
			FLOATI	Integer*2	Real
			FLOATJ	Integer*4	Real

(cont.)

8. [x] is defined as the largest integer whose magnitude does not exceed the magnitude of x and whose sign is the same as that of x. For example [5.7] equals 5. and [−5.7] equals −5.

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Conversion to double-precision	1	DBLE	DBLE	Real	Double
			—	Double	Double
			—	Complex	Double
			DFLOTI	Integer*2	Double
			DFLOTJ	Integer*4	Double
Real part of complex or conversion to single-precision	1	REAL	REAL	Complex	Real
			FLOATI	Integer*2	Real
			FLOATJ	Integer*4	Real
			SNGL	Real	Real
			SNGL	Double	Real
Imaginary part of complex	1	—	AIMAG	Complex	Real
Complex from two reals	2	—	CMPLX	Real	Complex
Conversion to complex or complex from two arguments		CMPLX	—	Integer*2	Complex
			—	Integer*4	Complex
			—	Real	Complex
			CMPLX	Real	Complex
			—	Double	Complex
			—	Complex	Complex
Complex conjugate (if a=(X,Y) CONJG (a)=(X,Y)	1	—	CONJG	Complex	Complex
Double product of reals a(1)*a(2)	2	—	DPROD	Real	Double

(cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Maximum max (a(1),a(2),...a(n)) (returns the maximum value from among the argument list; there must be at least two arguments)	n	MAX	AMAX1	Real	Real
			DMAX1	Double	Double
			IMAX0	Integer*2	Integer*2
			JMAX0	Integer*4	Integer*4
		MAX0	IMAX0	Integer*2	Integer*2
			JMAX0	Integer*4	Integer*4
		MAX1	IMAX1	Real	Integer*2
			JMAX1	Real	Integer*4
		AMXA0	AIMAX0	Integer*2	Real
			AJMAX0	Integer*4	Real
Minimum min(a(1),a(2),..a(n)) (returns the minimum value among the argument list; there must be at least two arguments)	n	MIN	AMIN1	Real	Real
			DMIN1	Double	Double
			IMIN0	Integer*2	Integer*2
			JMIN0	Integer*4	Integer*4
		MIN0	IMIN0	Integer*2	Integer*2
			JMIN0	Integer*4	Integer*4
		MIN1	IMIN1	Real	Integer*2
			JMIN1	Real	Integer*4
		AMIN0	AIMIN0	Integer*2	Real
			AJMIN0	Integer*4	Real

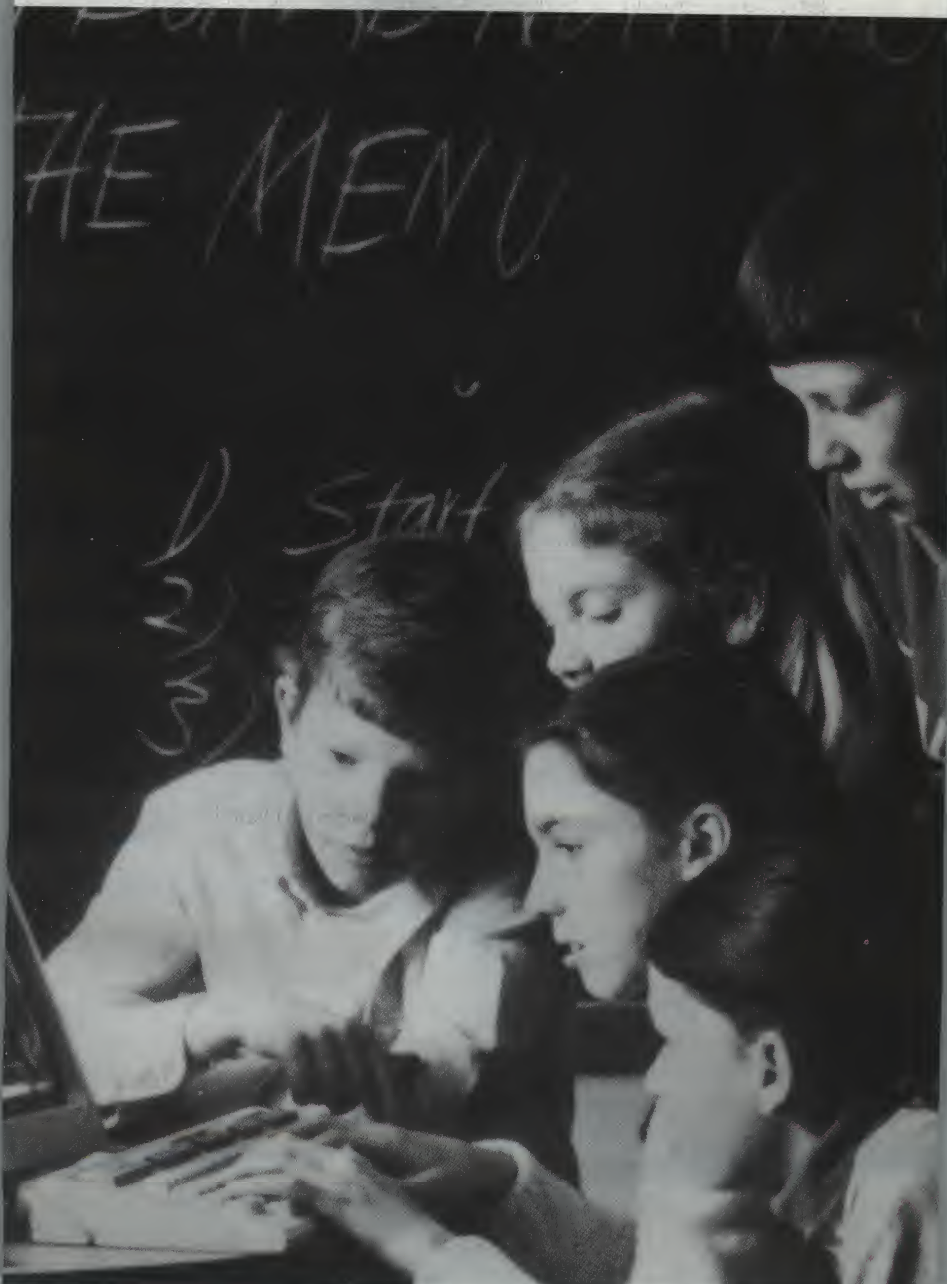
(cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Positive difference a(1) – (min(a(1,,a(2))) (returns the first argument minus the minimum of the two arguments)	2	DIM	DIM	Real	Real
			DDIM	Double	Double
			IIDIM	Integer*2	Integer*2
			JIDIM	Integer*4	Integer*4
		IDIM	IIDIM	Integer*2	Integer*2
			JIDIM	Integer*4	Integer*4
Remainder a(1) – a(2)* [a(1)/a(2)] (returns the remainder when the first argument is divided by the second)	2	MOD	AMOD	Real	Real
			DMOD	Double	Double
			IMOD	Integer*2	Integer*2
			JMOD	Integer*4	Integer*4
Transfer of sign | a(1)*Sign a(2)	2	SIGN	SIGN	Real	Real
			DSIGN	Doubler	Double
			IISIGN	Integer*2	Integer*2
			JISIGN	Integer*4	Integer*4
		ISIGN	IISIGN	Integer*2	Integer*2
			JISIGN	Integer*4	Integer*4
Bitwise AND (performs a logical AND on corresponding bits)	2	IAND	IIAND	Integer*2	Integer*2
			JIAND	Integer*4	Integer*4
Bitwise OR (performs an inclusive OR on corresponding bits)	2	IOR	IIOR	Integer*2	Integer*2
			JIOR	Integer*4	Integer*4
Bitwise exclusive OR (performs an exclusive OR on corresponding bits)	2	IEOR	IIEOR	Integer*2	Integer*2
			JIEOR	Integer*4	Integer*4

(cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Bitwise complement (complements each bit)	1	NOT	INOT JNOT	Integer*2 Integer*4	Integer*2 Integer*4
Bitwise shift (a(1) logically shifted left a(2) bits	2	ISHFT	IISHFT JISHFT	Integer*2 Integer*4	Integer*2 Integer*4
Random number (returns the next number from a sequence of pseudo-random numbers of uniform distribution over the range 0 to 1)	1 2	— —	RAN RAN	Integer*4 Integer*2	Real Real
Length (returns length of the character expression)	1	—	LEN	Character	Integer*2
Index (C(1),C(2)) (returns the position of the substring c(2) in the character expression c(1))	2	—	INDEX	Character	Integer*2
ASCII Value (returns the ASCII value of the argument; the argument must be a character expression that has a length of 1)	1	—	ICHAR	Character	Integer*2
Character relationals (ASCII collating sequence)	2 2 2 2	— — — —	LLT LLE LGT LGE	Character Character Character Character	Logical*2 Logical*2 Logical*2 Logical*2

Chapter 13 • BASIC



BASIC Is the Most Widely Used Language in the World Today

BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code. BASIC was developed at Dartmouth College to answer the need for an easy-to-learn, conversational programming language accessible to people who are not computer specialists. Characteristics of the BASIC language include simple English words, understandable abbreviations, and the familiar symbols for mathematical and logical operations.

BASIC is the most widely implemented and most widely used programming language in the world today. Digital's BASIC language implementations have always been in the vanguard of the computer industry, and today they are acknowledged as the industry leaders.

BASIC, in its interactive versions, like BASIC-PLUS gives novice programmers almost immediate use of the computer, allowing them to get results for mathematical requests very easily. In addition, with little training beginners can write and run meaningful programs. Interactivity encourages new users to practice and experiment with the language because it provides a quick response from the computer telling whether the instruction worked and, if it didn't, what went wrong. BASIC also includes powerful capabilities necessary to users who want to do file management, matrix manipulation, editing, and other more advanced computer operations.

Most of Digital's versions of BASIC offer the advantages of interactive program development, including powerful statement editing features plus HELP and debugging facilities. Their friendly environments have made BASIC one of the most popular programming languages in both commercial and technical applications, along with its traditional preeminence in academia.

PDP-11 BASIC Implementations

The following paragraphs give a brief overview of Digital's two BASIC implementations.

BASIC-PLUS-2

PDP-11 BASIC-PLUS-2 is the most powerful, most advanced BASIC language implementation available on PDP-11 systems. As a true compiler, BASIC-PLUS-2 significantly improves the performance of compute-bound BASIC applications. Fast program execution and a variety of advanced programming features make BASIC-PLUS-2 a highly productive programming environment and powerful enough for a wide variety of applications. BASIC-PLUS-2 is available on the RSX-11M-PLUS, RSX-11M, Micro/RSX, RSTS/E, and Micro/RSTS operating systems and is generally a superset of BASIC-PLUS and a subset of VAX-11 BASIC. This makes BASIC-PLUS-2 applications highly transportable across a wide variety of Digital systems. More detailed information on BASIC-PLUS-2 is included in the Product Descriptions section of this chapter.

BASIC-PLUS

BASIC-PLUS was specifically designed for and runs on RSTS/E, Micro/RSTS, RT-11, and RT-11 on Digital's Professional series 300 hardware.

Experienced programmers can use BASIC-PLUS's advanced features and facilities to produce complex and efficient programs. In general, BASIC-PLUS provides the following advantages:

- Programs can be written to conserve memory space and reduce execution time.
- Programs can handle a wide range of data by manipulating character strings.
- Programmers can obtain greater precision than is possible with floating-point and integer operands by using arithmetic and numeric string data manipulation.

It's not surprising that BASIC-PLUS is widely used for sophisticated scientific and business applications. Beginning programmers find BASIC-PLUS convenient and easy to use. BASIC-PLUS is also widely used as an educational tool in institutions ranging from elementary schools to universities.

Digital's Versions Share Common Features

Both BASIC implementations share syntax, language elements, and file structure features. In addition, programs in all three implementations are created, modified, and executed in similar fashion.

General Syntax

A BASIC program is composed of groups of statements containing instructions to the computer. Each group begins with a number that identifies it as a statement and indicates the order of statement execution relative to other lines in the program. Each statement starts with an English word specifying the type of operation to be performed.

More than one statement can be written on a single line when each statement after the first is preceded by a backslash. For example,

```
10 INPUT A,B,C
```

is a single statement line, whereas

```
20 LET X = 11 \PRINT X,Y,Z \IF X = A THEN 10
```

is a multiple statement line containing three statements—

LET, PRINT, and IF.

BASIC Language Elements

In addition to real and integer formats, BASIC accepts exponential notation. Numeric data can be input in any one or all of these formats. BASIC automatically uses the most efficient format for printing a number, according to its size. It automatically suppresses leading and trailing zeros in integer and decimal numbers, and formats all exponential numbers.

It can also process information in the form of strings. A *string* is a sequence of alphabetic, numeric, or special characters treated as a unit, either a constant or a variable.

A string constant is a list of characters enclosed in quotes that can be used in such diverse BASIC statements as PRINT, CALL, and CHAIN. String constants can also be used to assign a value to a string variable, for example, in the LET and INPUT statements, as with

```
30 LET A$ = "HELLO"
```

Subscripted variables provide additional computing capabilities for dealing with lists, tables, matrices, or any set of related variables. In BASIC, variables are allowed either one or two subscripts. For example, a list of floating-point values might be stored in an array A(I) where I goes from 0 to 5:

A(0), A(1), A(2), A(3), A(4), A(5)

This allows reference to each of the six elements in the list, and can be considered a one-dimensional algebraic matrix. Analogously, you can construct two-dimensional arrays by using two subscripts. For example,

B(I,J)

where I goes from 0 to 3 and J goes from 0 to 5, defines a 24-element matrix.

Any variable name followed by a percent sign (%) indicates an integer variable. For example, A%, C7%, C%(5).

Any variable name followed by a dollar sign ($) indicates a string variable (for example, A$, C7$), while a matrix variable name followed by the dollar sign denotes the string form of that variable (for example, V$(n), M2$(n), C$(m,n), G1$(m,n)).

Variables without % or $ suffixes are considered floating-point variables; for example, A, B7, C(I), D(J,K).

The user can assign values to variables by using a LET statement, by entering the value as data in an INPUT statement, or by using a READ statement with associated data statements. Values assigned to a variable do not change until the next time a statement that contains a new value for that variable is encountered.

Operators

BASIC performs addition, subtraction, multiplication, division, and exponentiation. IF-THEN statements have access to a variety of relational operators (less than, not equal, greater than, or equal to, for example). Most operators of both kinds work with strings as well as with numerical arguments; for strings, the relational operators do alphabetic comparisons.

Statements

The following summary of BASIC statements gives a brief explanation of each statement's use.

CALL	Transfers control to a subprogram, optionally passes parameters to it, and stores the location of the calling program for an eventual return.
CHAIN	Terminates execution of the program, loads the program specified, and begins execution of the lowest line number or, when a line number is present in the statement, at the specified line number.
CLOSE	Closes the file(s) associated with the logical unit number(s) and virtual file logical unit number(s).
DATA	Creates a data block for the READ statement. Can contain any combination of strings and numbers.
DEF FN	Defines a user function.
DIM	Reserves space in memory for arrays according to the subscripts specified.
END	Placed at the end of the physical end of the program to terminate execution (optional).
FOR	Sets up a loop to be executed a specified number of times.

GOSUB	Unconditionally transfers control to specified line of subroutine.
GOTO	Unconditionally transfers control to specified line number.
IF	Conditionally executes the specified statements or transfers control to the specified line number. If the condition is not satisfied, execution continues at the next sequential line. The expressions and the relational operator must be all string or all numeric.
INPUT	Inputs data from a file or from the user's terminal. Variables can be arithmetic or string.
KILL	Deletes the specified file.
LET	Assigns the value of an expression to the specified variable(s).
NAME AS	Renames the specified file.
NEXT	Placed at the end of the FOR loop to return control to the FOR statement.
ON GOSUB	Conditionally transfers control to the subroutine at one line number specified in the list. The value of the expression determines the line number to which control is transferred.
ON GOTO	Conditionally transfers control to one line number in the specified list. The value of the expression determines the line number to which control is transferred.
OPEN FOR INPUT [OUTPUT] AS FILE #n	Opens a file for input [or output] and associates the file with the specified logical unit number or channel number.
PRINT	Prints the values of the specified expressions on the terminal or, when specified, to the file associated with the logical unit expression. The TAB function can also be included.
PRINT USING	Generates output formatted according to a format string (either numeric or string).
RANDOMIZE	Causes the random number generator (RND function) to produce random numbers every time the program is run.
READ	Assigns values listed in DATA statements to the specified variables. These variables can be numeric or string.

REM	Contains explanatory comments in a BASIC program.
RETURN	Terminates a subroutine and returns control to the statement following the last executed GOSUB statement.
RESTORE	Resets to the beginning the data pointer.
STOP	Suspends execution of the program.

Arithmetic Functions

BASIC provides a variety of functions to perform mathematical operations.

ABS	Returns the absolute value of an expression.
ATN	Returns the arc tangent as an angle in radians.
COS	Returns the cosine of an expression in radians.
EXP	Returns the value of the constant e (approximately 2.71828) raised to a given power, which can be an expression.
INT	Returns the greatest integer less than or equal to a given expression.
LOG	Returns the natural logarithm of an expression.
LOG10	Returns the base 10 logarithm of an expression.
PI	Returns the value of pi (3.141593 approximately).
RND	Returns a random number between 0 and 1.
SGN	Returns value indicating the sign of an expression.
SIN	Returns the sine of an expression in radians.
SQR	Returns the square root of an expression.
TAB	Causes the terminal printhead to tab to column number given by an expression (valid only in PRINT).
SYS	Special system function calls; controls terminal I/O and performs special functions.

String Functions

CHR$	Generates a one-character string whose ASCII value is the low-order eight bits of the integer value of the given expression.
TIME$	Returns the time as a string.
DATE$	Returns the date as a string.
LEN	Returns the number of characters in the given string.
POS	Searches for and returns the position of the first occurrence of a substring in a string.

SEG$/MID	Returns the string of characters in the given positions in the string.
STR$	Returns the string that represents the numeric value of the given expression.
TRM$	Returns the given string without trailing blanks.
VAL	Returns the value of the decimal number contained in the given string expression.

User-Defined Functions

In some programs it may be necessary to execute the same sequence of statements in several different places. BASIC allows definition of unique operations or expressions and the calling of these functions in the same way as, for example, the square root or trigonometric functions. Each function is defined once and can appear anywhere in the program. User-defined functions simplify program entry, contribute to modular coding, and help to streamline programs.

BASIC Files

Data is stored either in sequential files or in random-access, virtual-array files. Data is read by an INPUT statement and written by a PRINT statement. Virtual arrays are random-access, disk-resident files similar to arrays stored in memory. A program can create and access virtual arrays just as it accesses memory-resident arrays — using array names and subscript values. Because the arrays are stored on disk, programmers can manipulate large amounts of data without affecting program size.

BASIC-PLUS-2 also provides the RMS-11 Record Management System. Through RMS-11, BASIC-PLUS-2 provides virtual array, block I/O, terminal-format, sequential, relative, and indexed files.

Creating, Modifying, and Executing BASIC Programs

A BASIC program is entered in the system using the editing commands. Once the program has been entered, it can be retrieved, listed, modified, or executed using the editing commands. These commands are

APPEND	Merges the program currently in memory with a program stored in a file. All lines in the program in memory that have duplicate line numbers with the program in the file are replaced by the lines from the program in the file.
BYE	Terminates the session at the terminal.
CLEAR	This command is used when a program has been executed and then edited. Before rerunning the program, the array and string buffers are cleared to provide more memory space (interpreters only).

LIST LISTNH	Types on the terminal the program currently in memory. A range of line numbers can be specified. The "NH" suffix suppresses header printing.
NEW	Clears the user area in memory and assigns a specified name to the current program. Used to create a new program.
OLD	Clears the user area and reads a program from a specified file into the user area in memory.
RENAME	Changes the current program name to a specified name.
REPLACE	Replaces the specified file with the program currently in memory.
RUN RUNNH	If issued with no file specification, executes the program currently in memory. If a file specification is issued, clears the user area, reads a program in from the file, and executes the program. The "NH" suffix suppress header printing. (Not in BASIC-PLUS-2/RSX.)
SAVE	Copies the contents of the user area to a file, lists the contents on the lineprinter, or punches the contents on paper tape.
UNSAVE	Deletes the specified file.

In addition to the editing commands, the BASIC system recognizes the following special control characters:

CTRL/C	Interrupts program execution and prints the READY message.
CTRL/O	Enables/disables console output.
CTRL/U	Deletes the current line being typed.

▪ BASIC-PLUS-2 Is Truly a Structured Language

BASIC-PLUS-2 Version 2 is the most powerful and advanced BASIC available on PDP-11 systems. It combines a powerful implementation language compiler with an integrated set of programs development utilities. Fast program execution is one way BASIC-PLUS-2 helps improve programmer productivity. A language-integrated I/O syntax conveniently accesses the RMS-11 record/file handling facilities. CALL statements allow modular structuring of programs. MAP statements permit variable-oriented data record access.

BASIC-PLUS-2 generates "threaded code." Threaded code produces smaller object programs than conventional inline instruction generation. Smaller programs mean fewer overlays are needed, and throughput is higher.

Language Features

BASIC-PLUS-2 language features provide greater programmer flexibility, more language statements, more sophisticated data typing and manipulation, and, through RMS, easier file access than most other BASIC implementations.

With the advent of Version 2, BASIC-PLUS-2 is now truly a structured language, allowing a "self-documenting" code that can be more quickly understood. It also has a built-in modularity that makes it easier to add new modules or modify old ones. This structured programming is supported in the following ways.

- *Up to 31 alphanumeric characters* can be used for variable and function names and statements labels, allowing for self-documenting names that indicate their use in in the program. These labels can also be used in place of line numbers in most parts of the program.
- *Block-structured statements* such as IF...THEN...ELSE...END...IF and SELECT...CASE...END SELECT allow programmers to directly implement structured code without resorting to makeshift shortcuts and excessive commenting.
- *User-named program constants* let programmers give meaningful names to often-used values in a program, such as integers that represent TRUE, FALSE, SUCCESS, and FAILURE flags.

Flexibility in program formatting allows programmers to arrange their source programs so that the functional blocks and the flow of the program can be easily identified.

Program Segmentation allows a program to be constructed of separately compiled modules headed by the SUB or FUNCTION statements to be subsequently accessed by the main module. Segmentation makes it easier to modify the program and gives programmers more flexibility in overlaying.

The CALL statement can pass parameters BY VALUE, REFerence, or DEScriptor.

The EXTERNAL statement provides access to global variables, functions, and constants, and allows data typing of parameters to aid in minimizing runtime mismatches.

The OTHERWISE clause ON GOTO and ON GOSUB provides a simple solution when the index expression is out of range.

Data Typing and Declarations

BASIC-PLUS-2 Version 2 has three general data types – *integer*, *floating point*, and *string*. There are further subdivisions within integer and floating-point that determine the storage requirements, range, and precision of the data type. In addition, there is a specialized data type called RFA that can contain only a record file address and is used with record file address I/O operations.

With these data types, the dollar sign ($) and percent sign (%) are usually unnecessary. While they are still used for undeclared strings and integers, they are not used for explicitly declared data types or with the DECLARE statement.

The DECLARE statement is used to explicitly declare variables, FUNCTIONS, CONSTANTS, and their data types. The DECLARE FUNCTION statement is used to define an internal function, including the function data type, number of arguments, and data type of the arguments. The DECLARE CONSTANT statement allows for the naming of constants and the assigning of a value to that constant.

Variables may be used in BASIC-PLUS-2 programs without being "declared" to the compiler. Scalar (that is, single-value) variables that are not declared are assigned storage from a general dynamic storage area available to the current program (or subprogram) only. Undeclared arrays have dimension size of 10, if one dimensional, or 10 by 10, if two dimensional. The DIMENSION statement is used to declare arrays with a size other than the default.

- COMMON AND MAP STATEMENTS

  The COMMON and MAP declarations are used to declare variables or arrays in a static, named storage area, accessible to other subroutines in the program image.

  The COMMON statement defines a named, shared area of memory called a COMMON block, occupied by specified variable values. These values can be read or changed by any BASIC-PLUS-2 subprogram with a COMMON block of the same name. The COMMON statement enables a subprogram to pass data to another program or subprogram. Strings passed in COMMON are of fixed length, thus reducing string handling overhead.

  COMMON and MAP are similar in function. The MAP statement allocates record buffer space. With MAP, users create a storage area that will serve as an I/O buffer associated with one or more open I/O channels. The MAP feature is unique to PDP-11 BASIC-PLUS-2 and VAX-11 BASIC—no other BASIC implementation provides it. MAP can save program space and perform better than other methods of declaring variables. Because a MAP statement defines fixed-length character strings, it provides maximum control over storage allocation and reduces overhead. And because MAP statements define data types at compiletime, execution time is faster.

- MAP DYNAMIC

  BASIC-PLUS-2 Version 2 also features a MAP DYNAMIC statement, which names the variables and arrays whose size and position in a MAP buffer can change at runtime. All pointers are set to the beginning of the MAP buffer by the MAP DYNAMIC statement. Once defined, the sizes and positions of these variables and arrays are changed with the REMAP statement.

Functions

A function performs one or more operations on a specified set of arguments and returns a result, either numeric or string, to the calling program. BASIC-PLUS-2 provides both library and user-defined functions.

- STRING HANDLING FUNCTIONS

With BASIC-PLUS-2, programmers can concentrate and compare strings; convert string and numeric representations; and analyze the composition of strings. BASIC-PLUS-2 includes string functions that

- Create a string containing a specified number of identical characters.
- Locate a substring within a longer string.
- Edit a string; change lowercase to uppercase; change square brackets to parentheses; trim trailing blanks/tabs from a string; delete form feeds, rubouts, line feeds, carriage returns, and nulls; trim parity, etc.
- Determine the length of a string.

BASIC-PLUS-2 also supports string functions to perform string-to-numeric and numeric-to-string conversions. Unlike many BASIC languages, BASIC-PLUS-2 imposes no limit on the size of string values or string elements of arrays manaipulated in memory, other than the amount of available memory.

- MATHEMATICAL AND NUMERIC STRING FUNCTIONS

BASIC-PLUS-2 provides algebraic, exponential, trigonometric, and random number functions. In addition, its string arithmetic functions permit greater precision — up to 56 digits — than floating-point calculations. BASIC-PLUS-2 also includes matrix functions for transposing and inverting matrices.

Program Control Constructs

In a BASIC program, control ordinarily moves from one line to another in consecutive line order. Within a line, control moves from statement to statement. However, execution can be diverted from the normal sequence to another portion of a program or to a subprogram, continue execution at that point, and then return control to the original program. BASIC-PLUS-2 provides a variety of methods to control program execution sequence. These include

- Subroutine constructs.
- CALL statement for subprograms.
- Statement modifiers.

Programmers can use BASIC-PLUS-2 to write structured programs much as they would use structured programming languages like Pascal. Structured programs are easier to write and maintain than conventional programs.

- SUBROUTINES

  A subroutine is a block of statements within a program that performs an operation and returns program control to the statement following the subroutine reference. The BASIC-PLUS-2 subroutine constructs use GOSUB, ON GOSUB, and RETURN statements.

  Subroutines differ from functions and subprograms. A subroutine is a sequence of instructions to be exectued several times in the course of a program. User-defined functions define a mathematical expression to be evaluated once, which then can be used repeatedly throughout a program. Subprograms are program units that can be separately compiled and then invoked from an external program.

- SUBPROGRAMS

  Separately compiled subprograms can be invoked using the CALL statement. Subprograms

  - Divide large programs into units that are more manageable.
  - Provide a convenient means for executing frequently used programs.
  - Permit control to transfer from one program to another.
  - Permit program overlays to conserve memory.

- STATEMENT MODIFIERS

  Programmers can use statement modifiers for conditional or repetitive execution of a statement. Modifiers save program text space and increase readability. Any nondeclarative statement in BASIC-PLUS-2 can have one of the five supported statement modifiers—FOR, IF, UNLESS, UNTIL, and WHILE.

  Modifiers cannot stand alone; they must be appended to a statement, and all executable BASIC-PLUS-2 statements can be modified.

  When using statement modifiers with the various forms of the IF statement, the following rules apply:

  - Append statement modifiers to either the THEN clause or the ELSE clause of an IF statement.
  - The statement modifier applies only to the clause it is appended to and not to the statement as a whole.

If more than one statement is on a line, the modifier applies only to the statement immediately preceding it. More than one statement modifier can be appended to a single statement. In this case, BASIC-PLUS-2 processes the modifiers from right to left.

Matrix Operations

With the MAT statement, the following operations can be performed on arrays:

- Assignment
- Addition
- Subtraction
- Multiplication
- Transposition
- Inversion

Each MAT operation statement begins with the keyword MAT followed by an expression to be evaluated. The value of one array can be assigned to another, for example, as in

```
MAT A = B
```

This statement sets each entry to array A equal to the corresponding entry of array B. A is redimensioned to the size of array B.

Files and Records

A major distinction between BASIC-PLUS-2 and BASIC-PLUS is access to the Record Management Services (RMS) with BASIC-PLUS-2. RMS greatly increases the ease with which programmers can develop and run complex programs.

There are four types of files in BASIC-PLUS-2:

- RMS record files.
- Terminal format files.
- Virtual array files.
- RSTS native (block I/O).

▪ RMS-11

RMS-11 is a record and file managment system that provides a variety of file organizations and access modes. Through RMS, BASIC-PLUS-2 provides virtual array, block I/O, terminal format, sequential, relative, and indexed files. This variety means users can choose file organizations and access methods best suited to individual applications.

BASIC-PLUS-2 has specific language elements for creating a file, describing the attributes of a file, opening a file, associating a record buffer with the file, describing the contents of the record buffer, performing input/output operations on a file, and allowing multiuser access to a file. With specific language elements for each of these operations, a CALL statement is not necessary for performing file handling functions. Because programmers can deal with logical records rather than with physical disk blocks, record and file handling is much easier than it is with other BASIC systems.

- TERMINAL FORMAT FILES
  A terminal format file is a stream of ASCII characters stored in lines of various lengths. The end of a line is determined by a line terminator, for example, line feed. BASIC-PLUS-2 stores these ASCII characters, including spaces and line terminators, exactly as they would appear on the terminal; hence the name terminal format file.

  Terminal format files are sequential access files (that is, they contain records that must be read or written one after another from the beginning of the file). This means that a record cannot be retrieved without first retrieving each of the items preceding it in turn.

  BASIC-PLUS-2 maintains a file pointer that keeps track of the user's location in the file. To add new items to an existing file without overwriting current information, the entire file must be read. This action places the file pointer at the end of the file where data can be added.

- VIRTUAL ARRAY FILES
  A virtual array file, like a terminal format file, is information stored on a disk. Once a virtual array file is opened, however, the similarity with terminal format files ends. Elements in a virtual array can be accessed exactly as elements in an array in memory.

  Virtual array files are random access files. The last element in a virtual array can be accessed as quickly as the first.

  When BASIC-PLUS-2 stores data in a virtual array file, it does not convert it to ASCII characters but rather stores it in the internal binary representation. Consequently, there is no loss of precision caused by data conversion.

- RSTS NATIVE (BLOCK I/O)
  Block I/O files make possible the most flexible and efficient technique of data transfer available under BASIC. Langauge extensions handle records composed of fixed-length fields, while disk and magnetic tape I/O operations read and write sequential data blocks. For disk files, the programmer can optionally specify a record or block number, thereby having complete random access to the file.

Programmers can dynamically define the record buffer area of a file as a series of fields and subfields. Alphanumeric data can be moved into a fixed-length field using either automatic truncation or padding (with spaces), depending on the relative lengths of the source and destination fields. Space in the record can be saved by packing the numeric fields using numeric–string conversion functions.

- CLUSTER LIBRARIES

  Cluster libraries allow you to write larger programs than you would otherwise be able to. This is accomplished by letting two or more resident libraries begin at the same address in a program. Although only one library in each cluster can be mapped at any given moment, a cluster library is automatically mapped whenever the program calls a subroutine in a library.

  Many Digital-supplied libraries of service routines and language routines, such as FMS and RMS, can be clustered. Users can also construct their own library clusters. Each cluster will have a default library plus one or more additional libraries that can take over the assigned set of virtual addresses.

▪ BASIC-PLUS Runs on RSTS/E, Micro/RSTS, and RT-11

The BASIC-PLUS language interpreter enables you to write programs in BASIC-PLUS and to interact with the operating system.

Functions and Features

BASIC-PLUS incorporates the following features:

- *Immediate Mode*: Commands can be executed immediately by BASIC-PLUS instead of being stored for later execution.
- *Program Editing*: An existing program can be edited by adding or deleting lines, or renaming the program. The user can combine two programs into a single program and request the listing of a program, either in whole or in part, on a terminal or lineprinter.
- *Program Control and Storage*: Facilities are included for storing both programs and data on any mass-storage device and retrieving them later for use during program execution.
- *Documentation and Debugging*: The insertion of remarks and comments within a program is easy in this version of BASIC. Program debugging is aided by the printing of meaningful diagnostic messages that pinpoint errors detected during the program's execution.

- *Access to System Peripheral Equipment*: The user is able to perform input and output with various equipment, such as disk, industry-compatible magnetic tape, lineprinters, and floppy disks.
- *Record I/O*: Provides a means of handling records composed of fixed-length fields in a highly efficient manner.
- *Matrix Computations*: A set of commands is available for performing matrix I/O, addition, subtraction, multiplication, inversion, and transposition.
- *Alphanumeric Strings*: Alphanumeric strings can be manipulated with the same ease as numeric data. Individual characters within these strings are accessible to the user.
- *Output Formatting*: The PRINT and PRINT-USING statements include facilities for tabs and spaces as well as precise specifications of the output line formatting and floating dollar sign, asterisk fill, and comma insertion in numeric output.
- *String Arithmetic*: A set of functions performs arithmetic on operands that are numeric strings instead of integer or floating-point numbers. This provides a means of calculating values of higher than normal precision, or values that must not be affected by round-off error (at the expense of slower execution time).

Immediate Mode Operation

Most BASIC-PLUS statements can either be included in a program for later execution or issued online at the terminal as commands to be immediately executed by the BASIC language processor. Immediate-mode operation is especially useful in two ways — it can be used to perform simple calculations that do not justify writing a complete program, and to debug a program.

To make program debugging easier, you can insert several STOP statements in the program. When the program is run, each STOP statement causes the program to halt and identify the line in the program at which the program was interrupted. You can then examine the current contents of variables, change them if necessary, and then continue.

Data Formats and Operations

BASIC-PLUS allows you to manipulate string, integer, numeric, or floating-point numeric data. BASIC-PLUS permits a user program to combine integer variables or integer-valued expressions using a logical operator to give a bitwise integer result. The logical operators AND, OR, NOT, XOR, IMP, and EQV operate on integer data in a bitwise manner.

BASIC-PLUS users working with floating-point numbers can increase accuracy of operations involving fractional numbers by using the scaled arithmetic feature or the string arithmetic functions. Futhermore, users can perform arithmetic operations using a mix of integer and floating-point numbers. If both operands of an arithmetic operation are either explicitly integer or explicitly floating point, the system automatically generates integer or floating-point results. If one operand is an integer and another is floating point, the system converts the integer to a floating-point representation and generates a floating-point result. If one operand is an integer and the other operand is a constant that can be interpreted either as a floating-point number or an integer, the system generates an integer result.

Matrix Manipulation

Variables of any type are legal in matrices, though any particular matrix must be composed of a single type of data. Explicitly dimensioning a matrix establishes the number of elements in each row and column and the number of elements in the matrix. (Implicitly dimensioned matrices are assumed to have 10 elements in each dimension referenced). Explicit dimensioning is done using the DIM statement.

By using the matrix I/O (MAT) statements, you can alter the number of elements in each row and the number of columns in the matrix, as long as the total number does not exceed the number defined when the matrix was dimensioned. The MAT operations do not set row zero or column zero, nor do they initailize values in the space allocated to the matrix unless specific MAT functions are executed.

The operations of addition, subtraction, and multiplication (including scalar multiplication) can be performed on matrices using the common BASIC mathematical operators; functions also exist for performing transposition and inversion of matrices.

Advanced Statement and Function Features

BASIC-PLUS extends the BASIC language by including several additional statements for easier logic flow control, function definitions, and faster response in a timesharing environment. The ON GOTO, ON GOSUB, IF-THEN-ELSE, FOR-WHILE, and FOR-UNTIL statements provide a variety of conditional controls over looping and subroutine execution. The ON ERROR GOTO statement allows the programmer to write subroutines that handle error conditions normally considered fatal. The program can test a system variable named ERR to determine which error occurred, and can examine a system variable named ERL to determine the line number at which the error occurred. SLEEP and WAIT allow program suspension, either for a specified time interval or until input from a terminal is received. The PRINT-USING statement provides special output formatting, including exponential representation, dollar signs, commas, trailing minus signs, and asterisk fill. The DEF statement allows multiple-line function definitions. Multiple-line function definitions can be nested, written in any data type, and contain any variety of argument types.

Five statement modifiers are available within BASIC-PLUS; IF, UNLESS, FOR (including FOR-WHILE and FOR-UNTIL), WHILE, and UNTIL. These modifiers are appended to program statements to indicate conditional execution of the statement or the creation of implied FOR loops.

BASIC-PLUS also includes several system functions and statements that allow program access to system information and conversion routines. The program can obtain the current date and time, the CPU time, connect time, kilocore ticks, and device time used for the job. It can convert a numeric value to a string date or time or vice versa, can swap bytes, or convert an integer in RADIX-50 format to a character string.

Table 13-1 ▪ BASIC Language Features—PDP-11

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Implementation Limits				
31-character names Up to 31 character variable names	—	X	X	X
Dynamic strings String length 32767 or maximum memory	—	X	X	X
Implicit continuation IF statements implicitly continued over multiple line	—	X	X	—
Labels Statement labels up to 31 characters	—	X	X	—
Maximum line 32767 Maximum line number allowed	—	X	X	X
Multiline statement Carry one statement across lines	—	X	X	X
Multistatement line Several statements on one line	—	X	X	X
Data Types, Constants, and Variables				
16-bit integer (WORD) Integer variables and constants	—	X	X	X
32-bit integer (LONG) 32-bit integers	—	X	X	—
8-bit integer (BYTE) Integer variables and constants	—	X	X	—
COMMON (X) I,J.. Defines a common data segment	—	X	X	—
CR, LF, FF, ESC... Predefined string constants	—	X	X	—
DATA 1,2,3,... Stores data for use by READ	X	X	X	X

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Data Types, Constants, and Variables (Cont.)				
DECLARE <type> Explicit data typing	—	X	X	—
DIM A(X [,Y]) Up to two dimensions	X	X	X	X
DIM A(X[,Y]) More than two dimensions	—	X	X	—
DOUBLE name Floating point variables and constants		X	X	X
ESC$ ASCII escape char: CHR$ (27)		*	*	—
EXTERNAL type External variables and functions	—	X	X	—
MAP (name) list Static data region <name>, has <list> variables	—	X	X	—
MAP DYNAMIC (name) Specifies fields to be allocated at runtime within name	—	X	X	—
PI Returns 3.14159...	X	X	X	X
Packed DECIMAL Data type, up to 31 digits precision	—	—	—	—
RECORD name User-defined data structure (like PASCAL or COBOL)	—	—	—	—
REMAP (name) <list> Runtime determination of I/O fields in a MAP DYNAMIC	—	X	X	—

*Product performs function, using different command.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Data Types, Constants, and Variables (Cont.)				
SINGLE real Floating point variables and constants	X	X	X	X
STRING A,B,... Declares a string variable	—	*	*	—
VARIANT (see RECORD), alternative field definition in a RECORD	—	—	—	—
Expressions and Assignments				
**Raise to a power		X	X	X
= = arith-op Approximate compare:numeric	—	X	X	X
= = string-op Exact string compare (trailing spaces count)	—	X	X	X
A,B,C = <exp> Multiple assignments in one statement	—	X	X	X
AND Logical operator	—	X	X	X
Arith-exp's + − * / ^	X	X	X	X
CONCATENATE "+" Joins string values	—	X	X	X
EQV Opposite of XOR	—	X	X	X
IMP 0 if first 1 and second argument = 0	—	X	X	X

*Function performed by product, uses different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Expressions and Assignments (Cont.)				
LET A = exp Optional, starts assignment statement	X[1]	X	X	X
LSET A$ = str-exp Left justifies data into random access field	—	X	X	X
NOT Logical negate	—	X	X	X
OR Logical operator	—	X	X	X
RSET A$ = str-exp Right justifies data into random access field	—	X	X	X
Relational-exp <> = < = > = <>	X	X	X	X
Stmt-modifiers Statement modifiers: FOR, IF, UNLESS, UNTIL, WHILE	—	X	X	X
XOR Exclusive OR — a logical operator	—	X	X	X
Trigonometric Functions				
ATN(X) Computes arc tangent	X	X	X	X
COS(X) Computes cosine	X	X	X	X
SIN(X) Computes sine, argument in radians	X	X	X	X
TAN(X) Computes tangent, argument in radians	X	X	X	X

[1]Not optional for ANSI Minimal.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Mathematic Functions				
ABS%(X) Returns integer, absolute value	–	X	X	–
ABS(X) Computes absolute value	X	X	X	X
COMP%(A$,B$) Compares two number-strings	–	X	X	X
DET Determinant from MAT INV	–	X	X	X
DIF$(A$,B$) String difference of two number-strings	–	X	X	X
EXP(X) Computes natural antilog	X	X	X	X
FIX(F) Truncates fractional part	–	X	X	X
INT(X) Largest integer not greater than argument	X	X	X	X
LOG(X) Natural log	X	X	X	X
LOG10(X) Returns base 10 log	–	X	X	X
MAG(X) Returns absolute value, with same data type as argument (ABS always returns floating)	–	X	X	–
NUM Last row of last MAT input	–	X	X	X

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Mathematic Functions (Cont.)				
NUM2				
Column number, last element of MAT input	–	X	X	X
PLACE$(S$,N)				
Precision of S$ set by n	–	X	X	X
PROD$(A$,B$,N)				
String product A$* B$	–	X	X	X
QUO$(A$,B$,N)				
String quotient	–	X	X	X
RND [(0)]				
Returns random number between 0 and 1	*	X	X	X
RND(N)				
Returns random number between 1 and argument	X	*	*	*
SGN(X)				
Returns the sign of a value	X	X	X	X
SQR(X)				
Computes square root	X	X	X	X
SUM$(A$,B$)				
String addition	–	X	X	X
String and Formatting Functions				
CVT$$(A$,N)				
String editing function	–	X	*	X
EDIT$				
String editing function	–	X	X	X
FORMAT$(X,A$)				
Expression –> formatted output string	–	X	X	–

*Product performs function, using different command.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
String and Formatting Functions (Cont.)				
INSTR(S,A$,B$) Returns starting position of a substring	—	X	X	X
LEFT$(A$,6) Returns left portion of a string	—	X	X	X
LEN(A$) Returns the number of characters in a string	—	X	X	X
MID$(A$,S,L) Returns substring within a string	—	X	X	X
NUM$(EXP) String = PRINT format	—	X	X	X
NUM1$(EXP) NUM$, without spaces	—	X	X	—
POS(A$,B$,P) Returns start index of B$ in A$, begins at column P	—	X	X	—
RIGHT$(A$,N) Returns right portion of a string	—	X	X	X
SEG$(A$,ST,ND) Extracts a substring from a string	—	X	X	X
SPACE$(X) Returns a string of X spaces	—	X	X	X
STR$(N) Converts a numeric value to a string	—	X	X	X
STRING$ Returns a string of n identical characters	—	X	X	X

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
String and Formatting Functions (Cont.)				
TRM$(S) Removes trailing blanks, tabs	–	X	X	X
VAL%(I) Integer value of string I	–	X	X	–
VAL(N$) Evaluates a string, returns a number	–	X	X	X
XLATE(S1,S2) Translate, S2 = table	–	X	X	X
Time and Date Functions				
CLK$ Returns HH:MM:SS time	–	*	*	*
DATE$ [(N)] Returns current date, 18-character string	–	X	X	X
TIME[$] (N) Returns time in 24-hour format string	–	X	X	X
Data Conversion Functions				
ASCII(A$) Code for character	–	X	X	X
CHR$(N) Returns ASCII character	–	X	X	X
CVT%(A$) Two characters -> integer	–	X	X	X
CVT$F(A$) Four characters—floating point	–	X	X	X

*Product performs function, using different command.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Time and Date Functions (Cont.)				
CVT%$(I%) Integer —> two characters	—	X	X	X
CVTF$(X) Floating value —> four characters	—	X	X	X
DECIMAL(X[,D,S]) Forces conversion to DECIMAL	—	—	—	—
INTEGER(X,BYTE), (X,WORD), (X,LONG) Forces conversion to INTEGER	—	X	X	—
OCT$(N) Computes octal value, returns string	—	—	—	X
RAD$(N) RAD-50 equivalent	—	X	X	X
REAL(X,SINGLE), (X,DOUBLE), (X,GFLOAT), (X,HFLOAT) Forces conversion to REAL	—	X	X	—
Input/Output Functions				
BUFSIZ(C) Returns buffersize for #c	—	X	X	X
CCPOS(N) Current character position in channel	—	X	X	X
FSP$(n) File description, channel #n	—	X	X	—
INKEY$ Gets keyboard character, if available		*	*	X

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Input/Output Functions (Cont.)				
MAGTAPE(FN,X,N) Magtape handling. F = function, X = number of records, N = channel	—	X	X	X
ONECHR(N) Starts 1-character input mode on #N	—	X	X	—
RECOUNT Length of last input	—	X	X	X
SPC(N) Prints a line of n spaces		*	*	*
SPEC%(...) RSTS peripheral control functions	—	X	—	X
STATUS Return host status/OPEN	—	X	X	X
Miscellaneous Functions				
FRE(A$) Returns amount of unused string space	—	*	*	—
FRE(X) Returns total amount of free memory space		*	*	—
LOC(X) Returns address of variable	—	—	—	—
MEM Returns the amount of free memory	—	*	*	—
PEEK(X) Returns memory contents at byte X (AKA:EXAM[INE])	—	X	—	X

*Product performs function, using different command. (cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Miscellaneous Functions (Cont.)				
RCTRLO Override effect of ^0, resume output	–	X	X	X
SWAP%(X) Transposes low two bytes	–	X	X	X
SYS (A$) Service call	–	X	–	X
User Definition of Subroutines and Functions				
DEF FN...FNEND Multistatement DEF...FNEND	–	X	X	X
DEF FNx(...) = exp Defines a user function	X	X	X	X
END DEF Define lexical end of a DEF	–	X	X	*
SUB NN(A,...,Z) Defines CALLable subroutine and arguments	–	X	X	–
Control Statements				
CALL name(args) Transfer control to a SUBROUTINE	–	X	X	–
CHAIN Load and execute specified program	–	X	X	X
END END program execution	X	X	X	X
END FUNCTION Delimits end of DEC BASIC FUNCTION	–	X	X	–

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Control Statements (Cont.)				
END IF Structured end of IF (vs. line number)	—	X	X	—
END SUB Defines lexical end of a SUB	—	X	X	—
EXIT Return to system level	—	—	—	X
EXIT Exit a labeled block	—	X	X	—
EXIT LABEL Structured BLOCK EXIT statement	—	X	X	—
FOR-NEXT Program loop	X	X	X	X
FUNCTION N(args) Separate FUNCTION subprogram	—	X	X	—
EXIT DEF Exit from DEC BASIC DEF	—	X	X	—
EXIT FUNCTION Exit from DEC BASIC FUNCTION	—	X	X	—
EXIT SUB Exit from DEC BASIC SUB	—	X	X	—
GOSUB NNNNN Transfers control to an internal subroutine	X	X	X	X
GOTO NNNNN Transfers control to a specific line number	X	X	X	X
IF <EXP>... Conditional statement	X	X	X	X

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

Control Statements (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
IF-THEN-ELSE Conditional statement with alternative	—	X	X	X
ITERATE <label> Structured equiv. of GOTO...NEXT in a loop		X	X	—
ON GOSUB..OTHER OTHERWISE out-of-range target for ON GOSUB		X	X	—
ON GOTO...OTHER OTHERWISE out-of-range target for ON GOTO		X	X	—
ON X GOTO/GOSUB Multiway branch with GOTO, GOSUB	X	X	X	X
RETURN Returns from a subroutine called by GOSUB	X	X	X	X
SELECT..CASE Structured CASE statement		X	X	—
STOP Stops execution of a program	X	X	X	X
SUBEXIT Subroutine EXIT statement	—	X	X	—
WHILE/END Delimits a WHILE repeat loop		*	*	*

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Event and Error Handling Statements				
CTRLC Enable ^C trapping	—	X	X	X
RCTRLC Disable ^C trapping	—	X	X	X
ERL Line number where error occurred	—	X	X	X
ERN$ Module name with error	—	X	X	—
ERR Code for most recent error	—	X	X	X
ERT$(N) Returns text string for ERR-n	—	X	X	X
EXTYPE HANDLER use: exception code	—	*	*	*
ON ERROR GOTO Establishes an error handler	—	X	X	X
ON ERROR GOTO 0 Disables error handling	—	X	X	X
ON..GOBACK Error exit to calling routine	—	X	X	—
RESUME [NNNNN] Ends an error handling routine	—	X	X	X

*Product performs function, using different command.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Matrix Manipulation Statements				
MAT A=CON(EXP) Assigns result of exp to elements of a	—	X	X	X
MAT A=IDN(M,N) Assigns identity matrix for (M x N) to al	—	X	X	X
MAT INPUT [#n]v.. Matrix INPUT	—	X	X	X
MAT LET al=expr Matrix assignment	—	X	X	X
MAT LINPUT [#]v.. Matrix string input	—	X	X	—
MAT PRINT [#N] v Matrix PRINT	—	X	X	X
MAT READ v,v... Matrix READ	—	X	X	X
MAT al = TRN(a2) Matrix transpose of a2, assigned to al	—	X	X	X
MAT al = ZER Sets elements of al to zero	—	X	X	X
MAT al$ = NUL$ Elements of al$ set to null string	—	X	X	—
MAT expressions MAT arrl = arr2[+, −,*] arr3	—	X	X	X

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Input/Output Statements				
Block I/O Block I/O (random access)	–	X	X	X
CLOSE [#N,...] CLOSE all files, or a single specified file	–	X	X	X
DECIMAL KEYS Indexed file access and update	–	–	–	–
DELETE Record deletion	–	X	X	–
ECHO #N Enable input echo on #n	–	*	*	*
FIELD #N,... Organizes a random file buffer into fields	–	X	X	X
FIND Record I/O FIND	–	X	X	–
GET Record GET	–	X	X	X[1]
INPUT "SS", list Prompts for INPUT with SS (any length literal)	–	X	X	X
INPUT #B, list INPUTs from disk unit b.	–	X	X	X
INPUT <list> INPUTs data from the keyboard	X	X	X	X
Integer Keys Indexed File access and update	–	X	X	–
KILL "filespec" Deletes a disk file	–	X	X	X

*Product performs function, using different command. (cont.)

[1]RSTS/E native files only.

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Input/Output Statements (Cont.)				
LINPUT #N, A$ Input a line from file #n	–	X	X	X
LINPUT ["S",]A$ Input a line, excluding terminating CR/LF/...	–	X	X	X
MOVE TO/FROM Item pack/unpack in record I/O	–	X	X	–
NAME A$ AS B$ Renames a data file	–	X	X	X
NOECHO #N Disable input echo, ch #n	–	*	*	*
OPEN ... Opens file, assigns mode and buffer	–	X	X	X
OPEN VIRTUAL Virtual (disk) array	–	X	X	*
PRINT #B,list Write data to sequential file buffer	–	X	X	X
PRINT #F USING.. Formatted sequential write to disk	–	X	X	X
PRINT TAB(N),... Move cursor right to specified TAB position	X	X	X	X
PRINT USING A$.. Prints formatted value(s) on screen	–	X	X	X
PRINT list Prints item(s) at current cursor position	X	X	X	X

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Input/Output Statements (Cont.)				
PUT				
Record PUT	—	X	X	X[1]
PUT #N				
Move data from file buffer to random access disk	—	X	X	X[1]
READ list				
READs value from a DATA statement	X	X	X	X
RESTORE				
Resets pointer to first item of first DATA	X	X	X	X
RESTORE #N				
Restore/rewind file		X	X	*
RFA Access				
Direct (record file address) access to indexed, sequential, or relative file	—	X	X	—
Record Locking				
Explicit record locking	—	X	X	X
SCRATCH				
Truncate a file	—	X	X	—
SLEEP N				
Wait n seconds, resume	—	X	X	X
Segmented Keys				
Indexed file access and update	—	X	X	—
UNSAVE filespec				
Deletes program on disk	—	*	*	X
UPDATE				
Record update	—	X	X	—
USEROPEN xxx				
Invoke xxx at OPEN to fill in file attributes	—	X	X	—

*Product performs function, using different command. (cont.)

[1]RSTS/E native files only.

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Input/Output Statements (Cont.)				
WAIT N Set n seconds input timeout	—	X	X	X
WIDTH NNN Sets printed line width for PRINT (or LPRINT)	—	—	—	*
Program Debugging Statements				
BREAK 100 Set break points in a program	—	X	X	X
CONT Continues execution after STOP/BREAK	—	X	X	X
TROFF Turns off the program trace function	—	—	—	*
TRON Turns on the program trace function	—	—	—	*
UNBREAK Remove breakpoint set by the BREAK command	—	X	X	X
Program Preparation Statements and Commands				
APPEND F Add file text to end of pro-gram in memory	—	X	X	X
AUTO [ST [,INC]] Numbers lines automatically	—	*	*	—
CAT [file] List file(s) available for use	—	*	*	X

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Program Preparation Statements and Commands (Cont.)				
COMPILE Compile program, do checks, don't RUN	—	X	X	X
Cross reference Compiler produces cross-reference listing	—	X	X	*
DELETE M [— N] Erases program lines from memory	—	X	X	X
EDIT Puts user into EDT with program loaded	—	—	—	—
EDIT [N] Puts computer into edit mode for specified line	—	X	X	—
ERASE Deletes an array (+ or a program). also see SCRATCH	—	*	*	*
Flag DECLINING Flag use of obsolete/non-transportable syntax	—	X	X	—
HELP/INQ Print HELP for topic	—	X	X	—
Immediate Mode Direct execution of unnumbered BASIC line	—	—	—	X
LIST List program lines to terminal	—	X	X	X
LIST M [— N] List program line(s) M [to N]	—	X	X	X

*Product performs function, using different command.

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Program Preparation Statements and Commands (Cont.)				
LISTNH List, print no heading	–	X	X	X
LOAD filespec Load precompiled program file from disk	–	X	X	–
NEW [name] Erase program from memory, initialize variables	–	X	X	X
OLD name Bring program from disk to memory	–	X	X	X
Opt/Line Nums Line numbers are generally optional		X	X	–
RENAME name New name for program in memory	–	X	X	X
REPLACE [name] Write program in memory onto disk	–	X	X	X
RESEQUENCE ... Renumbers a program also see RENUMBER	–	*	*	*
RUN Execute resident program	–	X	X	X
RUN file Loads and executes specified disk program	–	X	X	X
RUNNH ... RUN, print no heading	–	X	X	X

*Product performs function, using different command.

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Program Preparation Statements and Commands (Cont.)				
SAVE [file] Saves the BASIC program on disk	—	X	X	X
SEQUENCE N [,I] Generate line numbers	—	X	X	—
SET xxxx Turn compiler/environment options on or off	—	X	X	—
Source Listing Compiler produces listings with source and error messages	—	X	X	—
Subset Flagger /BP2 puts you in BASIC-PLUS-2 subset of VAX BASIC	—	—	—	—
ANSI flagger /ANSI checks ANSI minimal conformance	—	—	—	—
Line-by-line Syntax Check /SYNTAX causes each line to be checked as it is entered	—	X	X	*
/VARIANT Allows conditional compilation on user supplied value				
Directives to the BASIC System				
%ABORT <text> (See %IP), stop compilation immediately	—	X	X	—
%CROSS Enable cross reference, in program listing	—	X	X	—
%IDENT <name> Program id, goes in .OBJ file as well as listing file	—	X	X	—

(cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Cont.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Directives to the BASIC System (Cont.)				
%IF <cond> %THEN Conditional compilation of source text	—	X	X	—
%INCLUDE — %CDD INCLUDE from a Common Data Dictionary	—	—	—	—
%INCLUDE Directive—inserts text from <file>	—	X	X	—
%LET Allows assignments to lexical variables	—	X	X	—
%LIST Directive—enables compiler listing	—	X	X	—
%NOCROSS Disable cross reference, in program listing	—	X	X	—
%NOLIST Directive—disables compiler listing	—	X	X	—
%SBTTL <text> Subtitle (line 2) for listing pages	—	X	X	—
%TITLE <text> Title text for line 1 of listing pages	—	X	X	—
%VARIANT Returns user-supplied /VARIANT value	—	X	X	—
BRLRES filespec Selects BASIC resident library	—	X	X	—

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Directives to the BASIC System (Cont.)				
CLEAR CLEARs all variables and program (SCRATCH)	—	*	*	X
Compiler Generates object code	—	X	X	—
DSKLIB F Selects disk library/link	—	X	X	—
IDENTIFY Print compiler ID line	—	X	X	*
LIBR filespec Select shared runtime library	—	X	X	—
LOCK/sw/sw.. Set BUILD/COM switches	—	X	X	—
ODLRMS filespec Select RMS library/link	—	X	X	—
RMSRES filespec Select RMS shared resident library	—	X	X	—
SCALE N Set floating point scale factor (0-6)	—	X	X	X
SCRATCH Reinitializes BASIC memory (see CLEAR, NEW also)		X	X	X
SHOW Print present defaults		X	X	X

*Product performs function, using different command. (cont.)

Table 13-1 ▪ BASIC Language Features—PDP-11 (Concl.)

	ANSI Minimal	BASIC -PLUS-2 RSTS	BASIC -PLUS-2 RSX	BASIC -PLUS
Miscellaneous Statements				
CHANGE X$ to Y String characters to array cells	—	X	X	X
CHANGE Y TO X$ Change array cells —> string characters	—	X	X	X
POKE adr, value Puts a value into RAM memory	—	*	*	X
RANDOM[IZE] (Re)seeds random number generator	X	X	X	X
REM Starts REMARK, goes to <eol>	X	X	X	X

*Product performs function, using different command.

Chapter 14 • COBOL

▪ COBOL-81 Offers Ease of Use for Business Applications

COBOL, the COmmon Business Oriented Language, is an industrywide data-processing language that has been designed specifically for business applications such as payroll, inventory control, and accounts receivable.

Digital's COBOL for the PDP-11 family is COBOL-81, Version 2, which is based on the ANSI-74 COBOL standard (X3.23-1974). It has been validated by the U.S. Government's Federal Compiler Testing Center. COBOL-81 runs on systems ranging in size from the PDP-11/23 and PDP-11/24 through the PDP-11/84. COBOL-81 is also available on Digital's Professional 300 series of personal computers when used in conjunction with the Professional Tool Kit.

COBOL-81 is a fully featured COBOL with mainframelike performance, and is designed for Micro/RSX, RSX-11M, RSX-11M-PLUS, RSTS/E business systems, and Professional 300 systems. It includes various Digital extensions to COBOL, including screen handling at the source language level.

COBOL-81 is a subset of VAX COBOL and shares a common syntax. Programs written for COBOL-81 can be compiled and executed using VAX COBOL without source changes, giving customers a migration path from the smallest PDP-11 systems to the largest VAX/VMS systems. This compatibility is important to the increasing number of customers who combine VAX and PDP-11 systems to meet their business needs.

VAX COBOL has a COBOL-81 subset flagger, ideal for those who use both VAX and PDP-11 systems. You can write, debug, and run code, using the performance and tools of a VAX, and then run your program through the COBOL-81 subset flagger. Once no error flags appear, you can move your source code and recompile it on a PDP-11.

For PDP-11 users who may eventually want to upgrade to a VAX system, COBOL-81 has a VAX COBOL flagger that points out any problems that may arise in migration to a VAX.

▪ COBOL-81 Makes Programming More Productive

COBOL-81 provides features aimed at making the COBOL programmer and the COBOL programs highly productive. COBOL programs and programmers often use batch-mode COBOL on large machines, but COBOL-81 was designed with interactive features familiar to Digital customers and small computer users. For example,

- Source code may be entered by using one of the many text editors available on the PDP-11. The Digital terminal format files created by these editors are also shorter than ANSI format files and help save disk space and compiletime processing — very critical features on systems with limited disk space and processor power.
- Debugging can be handled interactively, allowing for error-free program development.
- The Build Overlay Description Language (BLDODL) utility offers a simplified way of structuring segmented programs or subprograms into efficient task images.

With COBOL-81, programmers have the tools needed for interactive video data processing on VT100 series and VT52 videoterminals, as well as the Professional 300 series of personal computers.

The DISPLAY statement positions the cursor at any row or column on the screen, converts any valid COBOL data type to a format suitable for screen display, and makes the terminal beep to alert the user. It also controls many of the VT100-series Advanced Video Option features such as bolding, blinking, underlining, and reverse video. It can erase the entire screen, the entire line, or everything from the cursor to the end of the line or the bottom of the screen.

The ACCEPT statement lets the programmer input information, define fixed-field widths, right- or left-justify data, and convert screen-format text into any valid COBOL data type. ACCEPT can prevent data from appearing on the screen for security purposes, helps specify default values, and helps define special function keys, and the auxiliary keypad. ACCEPT controls the same VT100 Advanced Video Option features as does the DISPLAY statement.

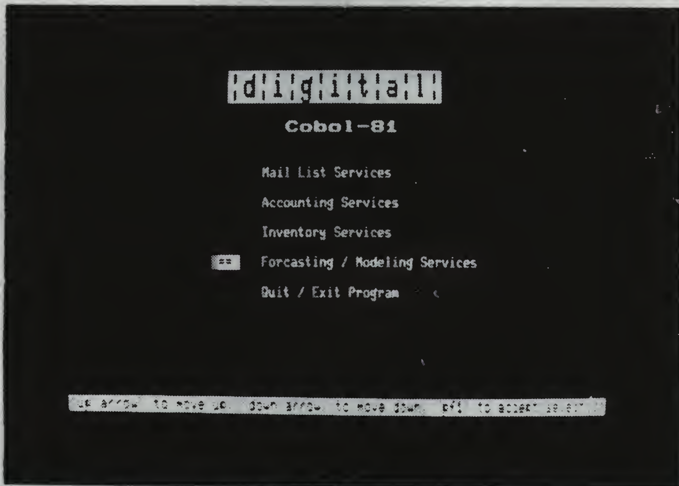
▪ COBOL-81 Measures Up to the Standards

COBOL-81 is ANSI-standard COBOL, rather than a small subset of the ANSI standard, found on other small systems. It also has features currently being recommended for the next ANSI COBOL standard, assuring future transportability of software.

COBOL-81 meets high-level requirements on many modules as described by FIPS PUB21-1. These modules and requirements are shown in Figure 14-1 and Table 14-1.

The COBOL-81 processing modules are

- Nucleus — contains all the essential language elements required for internal processing.
- Table Handling Module — provides the ability to define and manipulate tabular data.
- Sequential I/O Module — provides the ability to define and access sequentially organized files.
- Indexed I/O Module — provides the ability to define and access indexed sequential files including dynamic access.
- Segmentation Module — allows for specifying overlay of the Procedure Division at object time.
- Library Module — provides the facility for copying predefined COBOL text into the source program.
- Interprogram Communication Module — provides the functionality of communicating with one or more other programs at runtime.
- SORT/MERGE Module — provides the facility for record and file sorting, and merging capabilities.
- Communication Module — provides the facility for system communications.



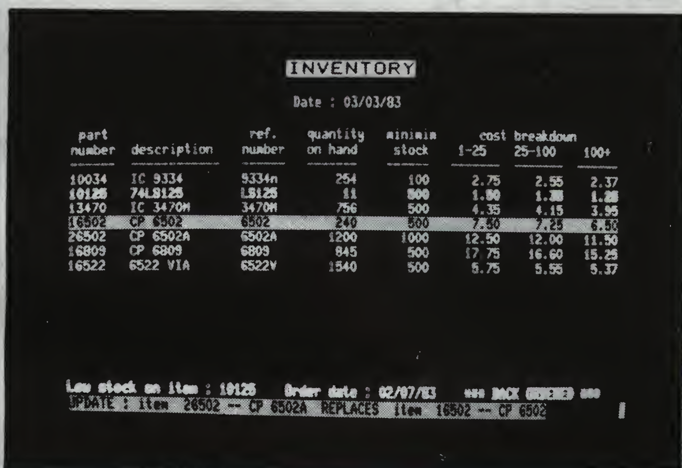


Figure 14-1 ▪ Screen Formats

Table 14-1 ▪ Language Features

ANSI Module	Level Supported By COBOL-81	FIPS PUB21-1 requirements for high level
Nucleus	2[1]	2
Table Handling	2	2
Sequential I/O	2	2
Relative I/O	2	2
Indexed I/O	2	2
Segmentation	2[2]	2
Library	1[3]	2
Debug	–[4]	2
Interprogram Communication	1	2
SORT/MERGE	2	2
Communication	—	2

[1] The nucleus module complies at Level 2, except that the ALTER statement and ALPHABET IS literal clause are not included.

[2] The segmentation module complies at Level 2, except that independent segments from Level 1 are not included.

[3] The library module includes a partial Level 2 REPLACING facility.

[4] COBOL-81 uses an interactive symbolic debugger, which may be substituted for the Debug module at all but the high level.

▪ COBOL-81 Can Be Installed by the Customer

COBOL-81 has been designed so that customers can install the software without requiring the aid of a Digital software specialist. The installation procedure determines a default compiler for the user's hardware configuration. If the compiler is acceptable to the user (indicated by a yes response to questions from the system), the default compiler is then built. If, however, the compiler is not acceptable, the system prompts the user with several more questions in order to build a customized compiler.

The compiler performance is impressive; it averages up to 500 lines per minute on a PDP-11/44. Because the compiler is designed for the Commercial Instruction Set (CIS), it generates compact high-performance object code, resulting in highly productive applications. This design also requires less use of time-consuming overlays.

▪ COBOL-81 Performs on the Benchmarks That Count

On a host of industry-standard benchmarks, COBOL-81 has shown its great speed. Typical of the results are the figures from USSTEEL, a nationally known, widely referenced synthetic benchmark developed by U.S. Steel Corporation. Tests run of USSTEEL on a PDP-11/44 system with the optional Commercial Instruction Set gave COBOL-81 a productivity index rivaling COBOL languages on machines of much greater size and cost.

COBOL-81 performs so well because its compiler was built for fast compilation and program execution. It takes advantage of the CIS hardware to enhance performance in data movement and packed-decimal arithmetic. Using CIS also results in memory-efficient object code and reduces the need for time-consuming overlays. Even without use of CIS, COBOL-81 performance is excellent.

COBOL-81 also supports cluster and resident libraries. These libraries decrease disk storage requirements for task images, increase the user's task area, increase taskbuilder performance, and increase memory availability in a multiuser environment.

▪ COBOL-81 Facilities Make Your Job Easier

COBOL-81 provides facilities for processing sequentail, relative, and indexed files. These include character string facilities, a CALL facility, SORT/MERGE, and library facilities.

Character String Facilities

COBOL-81 provides INSPECT, STRING, and UNSTRING verbs for easy handling of textual data. Using these verbs, programmers can count and/or replace embedded character strings and can join together or break out separate strings with various delimiters.

CALL Facility

The CALL facility allows COBOL programs to invoke separately compiled subprograms written in COBOL-81 or MACRO-11 Assembly Language. The CALL statement is used to execute routines that are external to the source file in which the CALL statement appears. This allows for greater flexibility of modular program development; permits functional specifications of small, well-defined source modules; and gives access to operating system-dependent features via subroutines written in MACRO-11.

The CALL statement facility has been extended by allowing the user to pass arguments BY REFERENCE (the default in COBOL) and BY DESCRIPTOR.

SORT/MERGE

The SORT/MERGE facility provides the SORT and MERGE verbs for record and file sorting and merging capabilities. Sorting can be specified by multiple keys either in ascending or descending order. Organizations of the input files can be totally independent.

Library Facility

COBOL-81 supports a full Level 1 ANSI-74 library facility. All frequently used data descriptions and program text sections can be stored in library files that are available to all programs. The library files can be copied at compiletime to reduce program preparation time and to eliminate a common source of error during program development.

▪ The Symbolic Interactive Debugger Allows Faster, Error-free Program Development

With COBOL-81's easy-to-learn, easy-to-use interactive debugger, programmers can debug programs by including the debugger when taskbuilding the program, rather than having to alter the source program during testing. Programmers can follow the program flow during the execution of a job. With the debugger, the programmer can

- Reference data items by their user-defined names.
- Reference section names and paragraph names.
- Examine and modify the value of variables during program execution.
- Optionally stop and restart programs at the line numbers, section name, or paragraph name specified by the programmer.
- Gain control at program commencement at abnormal termination.

Other Debugging Features

To make program debugging easier, the COBOL-81 compiler produces source listings with embedded English-language diagnostics. Fully descriptive diagnostic messages are listed at the point of error. Many error conditions, varying from simple warnings to fatal error detections, are checked at compiletime.

When an error occurs during execution, the type of error, program-name, and line-number of the source statement that caused the error are displayed on the user's terminal. If the program is executing with an active PERFORM, the line numbers of the active PERFORM statements will be produced on the user's terminal. When the error is detected during execution of a subprogram, a backward trace of the calling programs that were active at the time of the error is produced on the user's terminal.

The compiler provides the option of specifying data division and procedure division allocation maps and a cross-reference listing. The cross-reference listing is a listing of all data names, procedure names, and the sourceline numbers of those program lines containing the definitions and references. For each name, a list of ordered sourceline numbers is displayed. Sourceline numbers for defined items are distinguished from sourceline numbers for referenced items.

Source Program Formats

The COBOL-81 compiler accepts source programs that are coded using either the ANSI-standard format files or shorter, easy-to-enter Digital terminal format files. Terminal format files are designed for use with interactive text editors. They eliminate the line numbers and identification fields and allow the user to enter horizontal tab characters and short text lines. Terminal format files also help save disk space and compiletime processing, very critical features on small systems with limited disk space and processor power.

Utility Programs

COBOL-81 provides the REFORMAT and BLDODL utilities to aid the user in data processing.

The REFORMAT utility reads COBOL source programs that are coded using Digital terminal format and converts the source statements to the ANSI standard format accepted by other COBOL compilers throughout the industry. It also has the option of accepting programs written in ANSI standard format and to convert the source statements to Digital terminal format. This offers the advantage of saving disk space and compiletime processing when a user is initially migrating from a non-Digital COBOL system to COBOL-81.

The BLDODL utility combines skeleton overlay description files generated by COBOL compilations into a single ODL file for use by the task builder. This utility allows the user a simplified way of structuring segmented programs or subprograms into an efficient task image.

Commercial Instruction Set (CIS)

COBOL-81 takes full advantage of the CIS to enhance performance in data movement and packed-decimal arithmetic. The utilization of CIS also provides the added benefit of compact object code, reducing the requirement for time-consuming overlays.

Table 14-2 ▪ COBOL Language Features

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
LANGUAGE CONCEPTS				
Character Set				
Characters used for words				
0,1,...,9 A,B,...,Z	1	NUC	X	X
- (hyphen or minus sign)				
a,b,...,z		EXT	—	X
Punctuation characters				
. " ( ) space or blank	1	NUC	X	X
=	1	NUC	X	X
, ;	2	NUC	X	X
' (single quote)		EXT	—	—
Characters used in arithmetic operations				
+ − * / **	2	NUC	X	X
Characters used in relations				
<>=	2	NUC	X	X
Characters used in editing				
B O + − CR DB * $. ,	1	NUC	X	X
/	1	NUC	X	X
Separators				
semicolon and comma not permitted	1	NUC	X	—
semicolon and comma allowed	2	NUC	X	X
	1	(ANS-82)		
character strings	1	NUC	X	X
COBOL words	1	NUC	X	X
maximum 30 characters			X	X
maximum 31 characters		EXT	—	—
user-defined words	1	NUC	X	X
cd-name	1	COM	X	—
condition-name	2	NUC	X	X
data-name (first character must be alphabetic)	1	NUC	X	—
alphabet-name	1	NUC		X
area-name		DBM	—	—
data-name (need not begin alphabetic)	2	NUC	X	X

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
LANGUAGE CONCEPTS				
Character Set (Cont.)				
file-name	1	SEQ	X	X
index-name	1	TBL	X	X
keepLIST-name		DBM	—	—
level-number	1	NUC	X	X
library-name	2	LIB	X	—
mnemonic-name	1	NUC	X	X
paragraph-name	1	NUC	X	X
program-name	1	NUC	X	X
realm-name		DBM	—	—
record-name	1	NUC	X	X
report-name	1	RPW	X	—
routine-name (may be omitted)	1	NUC	X	—
schema-name		DBM	—	—
section-name	1	NUC	X	X
segment number	1	SEG	X	X
set-name		DBM	—	—
subschema-name		DBM	—	—
symbolic character				—
text-name	1	LIB	X	X
system names	1	NUC	X	X
computer-name	1	NUC	X	X
implementor name	1	NUC	X	X
language-name (may be omitted)	1	NUC	X	—
reserved words	1	NUC	X	X
key words	1	NUC	X	X
optional words	1	NUC	X	X
Connectives				
qualifiers: OF IN	2	NUC	X	X
series: comma, semicolon	2	NUC	X	X
logical: AND, OR, AND NOT, OR NOT	2	NUC	X	X
Special registers				
LINE-COUNTER, PAGE-COUNTER	1	RPW	X	—
LINAGE-COUNTER	2	SEQ	X	X

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
LANGUAGE CONCEPTS				
Character Set (Cont.)				
DEBUG-ITEM	1	DEB	X	—
TALLY	1	NUC	—	—
DB-CONDITION		DBM	—	—
DB-CURRENT-RECORD-ID		DBM	—	—
DB-CURRENT-RECORD-NAME		DBM	—	—
RMS-STS		EXT	—	X
RMS-STV		EXT	—	X
RMS-FILENAME		EXT	—	—
Figurative constants				
ZERO	1	NUC	X	X
ZEROS, ZEROES	2	NUC	X	X
SPACE	1	NUC	X	X
SPACES	2	NUC	X	X
HIGH-VALUE, LOW-VALUE	1	NUC	X	X
HIGH-VALUES, LOW-VALUES	2	NUC	X	X
QUOTE	1	NUC	X	X
QUOTES	2	NUC	X	X
ALL literal	2	NUC	X	X
Special character words				
Arithmetic operators	2	NUC	X	X
Relation operators	2	NUC	X	X
Literals	1	NUC	X	X
Nonnumeric literals have lengths from 1 through				
at least 120 chars	1	NUC	X	X
at least 256 chars		EXT	—	X
Quote character within non-numeric literals	1	NUC	X	X
Numeric literals have lengths from 1 through at least 18 digits			X	X
PICTURE character strings	1	NUC	X	X
Comment entries	1	NUC	X	X
Picture character string up to 255 characters		EXT	—	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
LANGUAGE CONCEPTS				
Character Set (Cont.)				
Qualification				
No qualification permitted	1	NUC	X	—
Qualification permitted	2	NUC	X	X
Subscripting to 3 levels	1	TBL	X	X
Subscripting to 48 levels	2	TBL	—	—
Indexing to 3 levels	1	TBL	X	X
Indexing to 48 levels	2	TBL	—	—
Source program structure				
Nested source programs	2	IPL	—	—
IDENTIFICATION DIVISION				
The PROGRAM-ID Paragraph	1	NUC	X	X
INITIAL clause	2	IPC	—	—
COMMON clause	2	IPC	—	—
The AUTHOR Paragraph	1	NUC	X	X
The INSTALLATION Paragraph	1	NUC	X	X
The DATE-WRITTEN Paragraph	1	NUC	X	X
The DATE-COMPILED Paragraph	2	NUC	X	X
The SECURITY Paragraph	1	NUC	X	X
The REMARKS Paragraph	1	NUC	—	—
ENVIRONMENT DIVISION				
Entire division may be omitted				X
Configuration Section				
Entire section may be omitted		EXT	—	X
The SOURCE-COMPUTER Paragraph				
computer-name	1	NUC	X	X
WITH DEBUGGING MODE phrase	1	DEB	X	—
The OBJECT-COMPUTER Paragraph				
computer-name	1	NUC	X	X
MEMORY-SIZE clause	1	NUC	X	X
PROGRAM-COLLATING SEQUENCE clause	1	NUC	X	X

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
ENVIRONMENT DIVISION				
Configuration Section (Cont.)				
SEGMENT-LIMIT clause	2	SEG	X	X
The SPECIAL-NAMES Paragraph				
Implementor-name IS mnemonic-name	1	NUC	X	X
ON STATUS	1	NUC	X	X
OFF STATUS	1	NUC	X	X
Implementor-name series	1	NUC	X	X
alphabet-name clause	1	NUC	X	X
STANDARD-1	1	NUC	X	X
STANDARD-2	1	NUC	—	—
NATIVE	1	NUC	X	X
Implementor-name	1	NUC	X	—
literal	2	NUC	X	—
CURRENCY-SIGN clause	1	NUC	X	X
DECIMAL-POINT clause	1	NUC	X	X
SYMBOLIC CHARACTER	2	NUC	—	—
ENVIRONMENT DIVISION				
Input-Output Section				
The FILE-CONTROL Paragraph				
SELECT	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	SRT	X	X
OPTIONAL	2	SEQ	X	X
ASSIGN-TO implementor name	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	SRT	X	X
MULTIPLE REEL/UNIT	1	SEQ	—	—
	1	SRT	—	—
RESERVE integer AREA(S)	2	SEQ	X	X
	2	REL	—	X
	2	INX	—	X
FILE-LIMIT literal THRU literal	1	SEQ	—	—
	1	REL	—	—
literal series	2	SEQ	—	—
	2	REL	—	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
ENVIRONMENT DIVISION				
Input-Output Section (Cont.)				
data-name THRU data-name	2	SEQ	—	—
	2	REL	—	—
data-name series	2	SEQ	—	—
	2	REL	—	—
Organization				
SEQUENTIAL	1	SEQ	X	X
RELATIVE	1	REL	X	X
INDEXED	1	INX	X	X
ACCESS MODE SEQUENTIAL	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
RANDOM	1	REL	X	X
	1	INX	X	X
DYNAMIC	2	REL	X	X
	2	INX	X	X
PROCESSING MODE SEQUENTIAL	1	SEQ	—	—
	1	REL	—	—
ACTUAL KEY	1	REL	—	—
RECORDING MODE clause	1	EXT	—	—
RELATIVE KEY	1	REL	X	X
RECORD KEY	1	INX	X	X
SYMBOLIC KEY		EXT	—	—
ALTERNATE RECORD KEY	2	INX	X	X
FILE STATUS clause	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
The I/O CONTROL Paragraph				
RERUN	1	SEQ	X	1[1]
	1	REL	X	1
	1	INX	X	1
SAME AREA	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X

[1]Supported for documentation purposes only.

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
ENVIRONMENT DIVISION				
Input-Output Section (Cont.)				
SAME RECORD AREA	2	SEQ	X	X
	2	REL	X	X
	2	INX	X	X
	2	SRT	X	X
SAME SORT/SORT MERGE AREA clause	2	SRT	X	X
SAME series	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
MULTIPLE FILE TAPES	2	SEQ	X	—
APPLY clause		EXT	—	X
Print-Control		EXT	—	X
Extension		EXT	—	X
Fill-Size		EXT	—	X
Mass-Insert		EXT	—	X
Contiguous		EXT	—	X
Window		EXT	—	X
DATA DIVISION				
Entire division may be omitted			—	X
Sections				
Communication section	1	COM	X	—
File section	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	SRT	X	X
	1	RPW	X	—
Linkage section	1	IPC	X	X
Working-Storage section	1	NUC	X	X
Report section	1	RPW	X	—
Subschema section		DBM	—	—
Schema section		DBM	—	—
Description Entries				
Communication description entry	1	COM	X	—
Data description entry	1	NUC	X	X

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
DATA DIVISION				
Description Entries (Cont.)				
File description entry	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	RPW	X	—
Record description entry	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
Sort-merge description entry	1	SRT	X	X
Report description entry	1	RPW	X	—
Report group description entry	1	RPW	X	—
Subschema description entry		DBM	—	—
Keeplist description entry		DBM	—	—
Clauses				
BLANK WHEN ZERO clause	1	NUC	X	X
BLOCK CONTAINS clause				
integer CHARACTERS/ RECORDS	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	RPW	X	—
ASCENDING/DESCENDING data-name	2	TBL	X	X
data-name series	2	TBL	X	X
INDEXED BY index-name	1	TBL	X	X
integer-1 TO integer-2 DEPENDING ON data-name	2	TBL	X	X
PAGE clause	1	RPW	X	—
PICTURE clause				
Character string may contain				
30 characters	1	NUC	X	X
255 characters		EXT	—	—
Data characters: A X 9	1	NUC	X	X
Operational symbols: S V P	1	NUC	X	X
Fixed insertion chars: 0 B , . $ + − CR DB	1	NUC	X	X
Fixed insertion char: /	1	NUC	X	X

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
DATA DIVISION				
Clauses (Cont.)				
Replacement or floating characters:				
$ + – Z *	1	NUC	X	X
Currency sign substitution	1	NUC	X	X
Decimal point substitution	1	NUC	X	X
RECORD CONTAINS clause	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	SRT	X	X
	1	RPW	X	—
RECORD VARYING clause		SEG	—	X
		REL	—	X
		INX	—	X
		SRT	—	X
REDEFINES clause				
must not be nested	1	NUC	X	—
may be nested	2	NUC	X	X
RENAMES clause	2	NUC	X	X
REPORT clause	1	RPW	X	—
SIGN clause	1	NUC	X	X
SOURCE clause	1	RPW	X	—
SUM clause	1	RPW	X	—
SYNCHRONIZED clause (SYNC)	1	NUC	X	X
TYPE clause	1	RPW	X	—
USAGE clause				
COMPUTATIONAL (COMP; means binary)	1	NUC	X	X
COMP-1 (floating point)		EXT	—	—
COMP-2 (double precision floating point		EXT	—	—
DISPLAY	1	NUC	X	X
DISPLAY-6 (SIXBIT)		EXT	—	—
DISPLAY-7 (ASCII)		EXT	—	—
DISPLAY-9 (EBCDIC)		EXT	—	—
INDEX	1	TBL	X	X
DATABASE-KEY		DBM	—	—
COMP-3 (packed decimal)		EXT	—	X

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
DATA DIVISION				
Clauses (Cont.)				
VALUE clause				
literal	1	NUC	X	X
literal series	2	NUC	X	X
literal THRU literal	2	NUC	X	X
literal range series	2	NUC	X	X
is EXTERNAL		EXT	—	—
is REFERENCE		EXT	—	—
VALUE of clause				
implementor-name IS literal	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	RPW	X	—
implementor-name IS data-name	2	SEQ	X	X
	2	REL	X	X
	2	INX	X	X
	2	RPW	X	X
PROCEDURE DIVISION				
Entire division may be omitted	1	NUC	—	X
USING phrase in Procedure Division header	1	IPC	X	X
Declaratives	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
	1	RPW	X	—
	1	DEB	X	—
Arithmetic expressions	2	NUC	X	X
Conditional expressions	1	NUC	X	X
Simple conditions	1	NUC	X	X
Relation condition	1	NUC	X	X
Relational operators				
[NOT] GREATER THAN	1	NUC	X	X
[NOT] >	2	NUC	X	X
[NOT] LESS THAN	1	NUC	X	X
[NOT] <	2	NUC	X	X
[NOT] EQUAL TO	1	NUC	X	X

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION (Cont.)				
[NOT] =	2	NUC	X	X
EQUALS		EXT	—	X
Comparison				
Numeric operands	1	NUC	X	X
Nonnumeric operands				
Must be equal size	1	NUC	X	—
May be unequal size	2	NUC	X	X
Class conditions	1	NUC	X	X
NOT option	1	NUC	X	X
Switch-status condition	1	NUC	X	X
Condition-name condition	2	NUC	X	X
Data Base condition		DBM	—	—
Success condition		EXT	X	X
Sign condition	2	NUC	X	X
NOT option	2	NUC	X	X
Complex conditions				
Logical operators AND OR and NOT	2	NUC	X	X
Negated simple conditions	2	NUC	X	X
Combined and negated combined conditions	2	NUC	X	X
Abbreviated combined relation condition	2	NUC	X	X
Statements				
Arithmetic statements				
Arithmetic operands limited to 18 digits	1	NUC	X	X
Overlapping operands	1	NUC	X	X
	1	TBL	X	X
Multiple results in arithmetic statements	2	NUC	X	X
ACCEPT statement				
Only one transfer of data	1	NUC	X	X
No restriction on number of transfers of data	2	NUC	X	—
AT END	1	EXT	—	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
END-ACCEPT	1	EXT	—	—
FROM	2	NUC	X	X
FROM DATE	2	NUC	X	X
FROM DAY	2	NUC	X	X
FROM DAY-OF-WEEK	2	NUC	—	—
FROM TIME	2	NUC	X	X
MESSAGE COUNT phrase	1	COM	X	—
ADD statement				
identifier/literal series	1	NUC	X	X
TO identifier	1	NUC	X	X
TO identifier series	2	NUC	X	X
GIVING identifier	1	NUC	X	X
GIVING identifier series	2	NUC	X	X
ROUNDED	1	NUC	X	X
SIZE ERROR	1	NUC	X	X
CORRESPONDING	2	NUC	X	X
END-ADD	1	NUC	—	—
ALTER statement				
procedure-name	1	NUC	X	—
procedure-name series	2	NUC	X	—
CALL statement				
literal	1	IPC	X	X
identifier	2	IPC	X	—
USING data-name	1	IPC	X	X
by value		EXT	—	—
by descriptor		EXT	—	X
by reference	2	IPC	—	X
by content	2	IPC	—	—
OMITTED argument		EXT	—	X
ON OVERFLOW	2	IPC	X	—
END-CALL	2	IPC	—	—
CANCEL statement	2	IPC	X	—

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
CLOSE statement				
single file-name	1	SEQ	X	X
file-name series	2	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
REEL	1	SEQ	X	X
UNIT	1	SEQ	X	X
NO REWIND	2	SEQ	X	
LOCK	2	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
FOR REMOVAL	2	SEQ	X	X
WITH DELETE		EXT	—	—
		DBM	—	—
COMMIT statement		DBM	—	—
COMPUTE statement Identifier	2	NUC	X	X
Identifier series	2	NUC	X	X
END-COMPUTE	1	NUC	—	—
CONNECT		DBM	—	—
CONTINUE	1	NUC	—	—
DELETE statement	1	REL	X	X
	1	INX	X	X
		DBM	—	—
END-DELETE	1	REL/INX	—	—
DISCONNECT		DBM	—	—
DISABLE statement				
INPUT	1	COM	X	—
TERMINAL	2	COM	X	—
OUTPUT	1	COM	X	—
KEY identifier/literal	1	COM	X	—
ENTER statement (may be omitted)	1	NUC	X	—
ENTRY statement		EXT	—	—
ERASE statement		DBM	—	—
EVALUATE	2	NUC	—	—
identifier/literal	2	NUC	—	—
arithmetic expression	2	NUC	—	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
conditional expression	2	NUC	—	—
TRUE/FALSE	2	NUC	—	—
when phrase	2	NUC	—	—
when other	2	NUC	—	—
END-EVALUATE	2	NUC	—	—
EXAMINE statement	1	NUC	—	—
EXIT statement	1	NUC	X	X
EXIT PROGRAM statement	1	IPC	X	X
FETCH statement		DBM	—	—
FIND statement		DBM	—	—
FREE statement	1	EXT/SEQ	—	—
	1	EXT/INX	—	—
	1	EXT/REL	—	—
		DBM	—	—
GENERATE statement		DBM	—	—
GET statement		DBM	—	—
GOBACK statement		EXT	—	—
GOTO statement				
TO optional	1	NUC	X	X
procedure-name required	1	NUC	X	X
procedure-name optional	2	NUC	X	—
DEPENDING ON phrase	1	NUC	X	X
IF statement				
Must be imperative statement	1	NUC	X	—
Nested statements	2	NUC	X	X
ELSE	1	NUC	X	X
		DBM	—	—
END-IF	1	NUC	—	—
INITIALIZE statement	2	NUC	—	—
identifier series	2	NUC	—	—
REPLACING	2	NUC	—	—
INITIATE statement	1	RPW	X	—
INSERT statement		DBM	—	—
	1	NUC	X	—

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
INSPECT statement				
single character data item	1	NUC	X	X
multicharacter data item	2	NUC	X	X
INVOKE statement		DBM	—	—
KEEP statement		DBM	—	—
MERGE statement	2	SRT	X	X
MODIFY statement		DBM	—	—
MOVE statement				
TO identifier	1	NUC	X	X
identifier series	1	NUC	X	X
CORRESPONDING	2	NUC	X	X
		DBM	—	—
MULTIPLY statement				
BY identifier	1	NUC	X	X
BY identifier series	2	NUC	X	X
GIVING identifier	1	NUC	X	X
GIVING identifier series	2	NUC	X	X
ROUNDED	1	NUC	X	X
SIZE ERROR	1	NUC	X	X
END-MULTIPLY statement	1	NUC	—	—
NOTE sentence	1	NUC	—	—
OPEN statement				
ALLOWING statement				
ALL	1	EXT	—	X
READERS	1	EXT	—	X
WRITERS	1	EXT	—	—
UPDATERS	1	EXT	—	—
NONE	1	EXT	—	—
READ	1	EXT	—	—
REWRITE	1	EXT	—	—
WRITE	1	EXT	—	—
DELETE	1	EXT	—	—
ANY VERB	1	EXT	—	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
INPUT				
Single file-name	1	SEQ	X	X
file-name series	2	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
REVERSED	2	SEQ	X	—
NO REWIND	2	SEQ	X	X
OUTPUT				
Single file-name	1	SEQ	X	X
file-name series	2	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
NO REWIND	2	SEQ	X	X
I-O				
Single file-name	1	SEQ	X	X
file-name series	2	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
EXTEND	2	SEQ	X	X
INPUT, OUTPUT, I-O and EXTEND series	2	SEQ	X	X
INPUT, OUTPUT and I-O series	1	SEQ	—	X
	1	REL	X	X
	1	INX	X	X
		DBM	—	—
PERFORM statement				
procedure-name	1	NUC	X	X
procedure-name optional	1	NUC	—	—
THRU	1	NUC	X	X
TIMES	1	NUC	X	X
UNTIL	2	NUC	X	X
VARYING	2	NUC	X	X
WITH TEST BEFORE/AFTER	1	NUC	—	—
END-PERFORM	1	NUC	—	—

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
READ statement				
file-name	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
INVALID KEY	1	REL	X	X
INTO identifier	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
NEXT	2	REL	X	X
AT END	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
KEY IS	2	INX	X	X
END-READ	1	SEQ/REL/INX	—	—
READY statement		DBM	—	—
RECEIVE statement				
MESSAGE	1	COM	X	—
SEGMENT	2	COM	X	—
INTO identifier	1	COM	X	—
NO DATA phrase	1	COM	X	—
RECONNECT statement		DBM	—	—
ROLLBACK statement		DBM	—	—
RELEASE statement				
record-name	1	SRT	X	X
FROM	1	SRT	X	X
REMOVE statement		DBM	—	—
RETURN statement				
file-name	1	SRT	X	X
INTO	1	SRT	X	X
AT END	1	SRT	X	X
REWRITE statement				
FROM identifier	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X

(cont.)

Table 14-2 ▪ COBOL Language Features (Cont.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
INVALID KEY phrase	1	REL	X	X
	1	INX	X	X
END-REWRITE	1	SEQ/REL/INX	—	—
SEARCH statement	2	TBL	X	X
SEEK statement	1	REL	—	—
SEND statement				
FROM identifier	2	COM	X	—
FROM identifier WITH	2	COM	X	—
WITH identifier	2	COM	X	—
WITH EGI	1	COM	X	—
WITH EMI	1	COM	X	—
WITH EPI		EXT	—	—
BEFORE/AFTER ADVANCING	1	COM	X	—
SET statement	1	TBL	X	X
SORT statement				
Limited to one SORT, -STOP and I-O procedures	1	SRT	X	—
Program not limited-to one SORT	2	SRT	X	X
COLLATING SEQUENCE phrase	2	SRT	X	X
START statement	2	REL	X	X
	2	INX	X	X
STOP statement	1	NUC	X	X
STORE statement		DBM	—	—
STRING statement	2	NUC	X	X
SUBTRACT statement				
identifier/literal series	1	NUC	X	X
FROM	1	NUC	X	X
FROM series	2	NUC	X	X
GIVING identifier	1	NUC	X	X
GIVING identifier series	2	NUX	X	X
ROUNDED	1	NUC	X	X
SIZE ERROR	1	NUC	X	X
CORRESPONDING	2	NUC	X	X
END-SUBTRACT	1	NUC	—	—

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
SUPPRESS statement	1	RPW	X	—
TERMINATE statement	1	RPW	X	—
TRACE statement		EXT	—	—
UNSTRING statement	2	NUC	X	X
USE statement		DBM	—	—
EXCEPTION/ERROR PROCEDURE				
ON file-name				
INPUT/OUTPUT/I-O	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
ON file-name series	2	SEQ	X	X
	2	REL	X	X
	2	INX	X	X
ON EXTEND	2	SEQ	X	X
LABEL PROCEDURE	2	SEQ	—	—
BEFORE REPORTING	1	RPW	X	—
GLOBAL PHRASE	2	IPC	—	—
USE FOR DEBUGGING statement				
procedure-name	1	DEB	X	—
procedure-name series	1	DEB	X	—
ALL PROCEDURES	1	DEB	X	—
ALL REFERENCES OF identifier	2	DEB	X	—
file-name series	2	DEB	X	—
cd-name series	2	DEB	X	—
WRITE statement				
record-name	1	SEQ	X	X
	1	REL	X	X
	1	SEQ	X	X
FROM identifier	1	SEQ	X	X
	1	REL	X	X
	1	INX	X	X
BEFORE/AFTER ADVANCING				
integer LINES	1	SEQ	X	X
identifier LINES	2	SEQ	X	X
mnemonic-name	2	SEQ	X	—

(cont.)

Table 14-2 ▪ COBOL Language Features (Concl.)

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
PROCEDURE DIVISION				
Statements (Cont.)				
PAGE	1	SEQ	X	X
AT END-OF-PAGE	2	SEQ	X	X
INVALID KEY	1	REL	X	X
	1	INX		X
END-WRITE	1	SEQ/REL/INX	—	—
SEGMENTATION				
segment-number (priority-number)	1	SEG	X	X
Fixed Memory Range 0-49	1	SEG	X	X
Non-fixed Memory Range 50-99	1	SEG	X	—
SEGMENT-LIMIT	2	SEG	X	X
LIBRARY				
COPY	1	LIB	X	X
text-name	1	LIB	X	X
literal		EXT	—	X
OF/IN LIBRARY	2	LIB	X	—
REPLACING	2	LIB	X	X
May appear anywhere a COBOL word may appear	1	LIB	X	X
Pseudo-text may be replaced	2	LIB	—	—
Identifier may be replaced	2	LIB	—	—
Literal may be replaced	2	LIB	—	X
word	2	LIB	—	—
from Dictionary		LIB	—	—
REFERENCE FORMAT				
Sequence Numbers	1	NUC	X	X
May be omitted		EXT	—	X
Area A	1	NUC	X	X
Division header	1	NUC	X	X
Section header	1	NUC	X	X
Paragraph header	1	NUC	X	X
Data Division entries	1	NUC	X	X
Area B	1	NUC	X	X

Features	Level	Implementation and Module Code	ANSI-74 X3.23	COBOL-81 V2
REFERENCE FORMAT (Cont.)				
Paragraphs	1	NUC	X	X
Data Division entries	1	NUC	X	X
Continuation of Lines				
Nonnumeric Literals	1	NUC	X	X
Words and Numeric Literals	2	NUC	X	X
Comments				
With *	1	NUC	X	X
With /	1	NUC	X	X
With D	1	DEB	X	—
With A-Z	1	EXT	—	—
FIPS INFORMATION				
FIPS Flagging at				
Low		FIP	—	—
Low-Intermediate		FIP	—	—
High-Intermediate		FIP	—	—
High		FIT	—	—
Certified at				
Low Level		FIP	—	—
Low-Intermediate		FIP	—	X
High-Intermediate		FIP	—	—
High		FIP	—	—

Chapter 15 • DIBOL-83

Digital's DIBOL Language Is Concise and Easy to Use

Digital's business-oriented programming language, DIBOL, was the first high-level commercial data processing language designed specifically to let application programs run in an interactive environment. Programs can do data manipulation, evaluation of arithmetic expressions, subscripting, record redefinition, calls to subroutines, and sequential, indexed, and random access to files. In addition, the interactive online debugging utility simplifies the programmer's job of isolating and correcting program errors.

Because of its high interactivity, DIBOL is a good program development language. Its simple syntax and free-form coding make the language easy to learn while providing for simpler program documentation.

DIBOL can be used in multijob timesharing environments to permit several application programs to run simultaneously. Of course, programs written for timesharing can also run on a single-user system. Another language feature is the availability of external subroutine libraries. Because subroutines can be held in libraries, programmers may create more compact programs, increasing efficiency and productivity. Such subroutines can be either Digital-supplied or user-developed.

DIBOL has simple, English-like statements which offer

- A full ASCII character set.
- Multilevel data access via file, record, field, and subfield.
- Subscripting.
- Array handling.
- Record overlays.
- Subroutine overlays.
- Predefined external DIBOL subroutines.
- Internal subroutines.
- Program sequence control (chaining) (ISAM).
- Rounding.
- File initialization and file labeling.
- Branching.

DIBOL Is a Structured Language

A DIBOL program is separated into two major parts — a Data Division and a Procedure Division. The Data Division contains the nonexecutable data specification statements that define the characteristics and identity of the data used by the program. The Procedure Division contains all the program statements that implement the actions or tasks to be performed. PROC and END statements are used to define the program structure. The PROC statement separates the Data Division statements from the Procedure Division statements, and the END statement specifies the logical end of the program.

DIBOL-83 Enriches the Language by Adding New Statements for Block Structure and Loop Control

DIBOL-83 is the newest version of DIBOL. This standard is the basis for a new expandable compiler that is bundled into every DIBOL product. The DIBOL-83 programming language is provided as part of the CTS-300 commercial operating system, which itself is based on the RT-11 operating system. It is also available as an option on RSX-11M-PLUS and RSTS/E, as well as VAX/VMS and P/OS, the operating system of the PRO 300 series of personal computers.

The DIBOL-83 compiler supports all the previous DIBOL statements and also the following set of statements that support structured programming concepts.

BEGIN-END	Allows a group of statements to appear wherever a single statement could appear.
DO-UNTIL FOR WHILE	Is used for loop control.
IF-THEN-ELSE USING	Allows conditional execution of one of a series of statements.

The DIBOL-83 compiler optimizes new or old DIBOL programs and reduces their size from 2 to 8 percent. This saves disk space and user memory. Old programs can be recompiled to take advantage of these newly introduced optimizations.

Statement Types Consist of English Language Verbs

A DIBOL statement has one or more elements. The first element is usually an English language verb that characterizes an action to be performed. The other elements of the statement are arguments, which consist of symbolic data names, references to statement labels, and expressions of data values or relationships. Arguments specify the objects of the action performed by the statement.

DIBOL statements fall into seven functional groups—compiler directives, compiler declarations, data specifications, data manipulation, control, intertask communications, and input/output. They include both arithmetic and logical expressions. The operators in an expression represent various arithmetic and manipulative functions of the DIBOL language. Operators are classified either as unary or binary operators and include most of the usual arithmetic, relational, and Boolean operations. In addition, there is a simple mechanism for formatting converted decimal data.

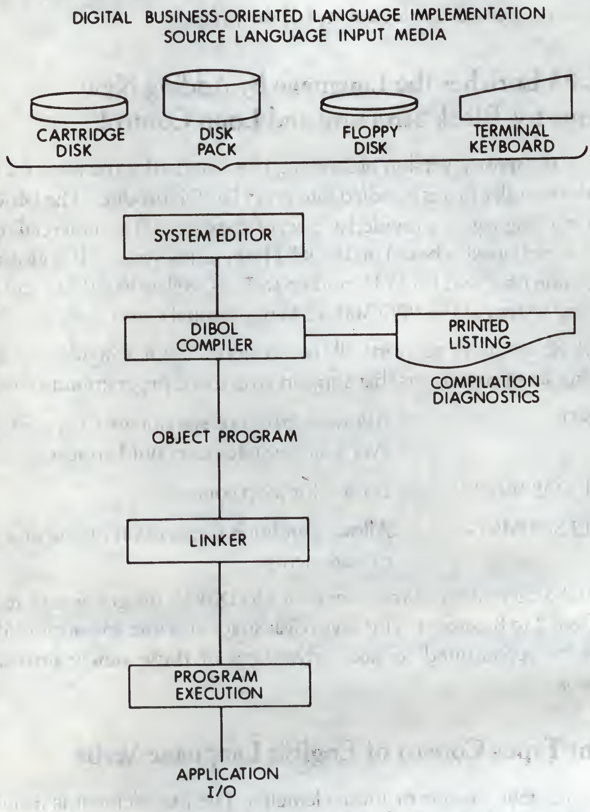


Figure 15-1 ▪ *DIBOL Implementation*

▪ Compiler Directive Statements – Compiler Declaration Statements

As the name suggests, these are instructions to the compiler; they are not executable at runtime and do not affect program operation. Among the compiler directives are statements that identify programs as external subroutines, that separate Data Division statements from Procedure Division statements, that cause the program listing to skip to a new page, that end the program, and that perform other, similar chores.

Compiler declaration statements provide information about the program structure to the compiler.

▪ Data Specification Statements

Data specification statements (also referred to as field definition statements) define and identify the characteristics of the data processed by a DIBOL program. For example, data can be either numeric or alphanumeric and can have certain size requirements and initial values. Fields of data that are grouped together are preceded by a RECORD or COMMON statement and may be redefined at that level. The data specification statements are

RECORD	Defines and identifies the beginning of one or more grouped fields.
COMMON	Defines and identifies the beginning of one or more grouped fields and allows external subroutines directly to use/share a field defined in the main program Data Division section.

▪ Data Manipulation Statements

These statements are used to perform calculations as well as data modification, conversion, and movement as stated below:

INCR	Increments a variable by one.
LOCASE	Converts uppercase characters to lowercase.
UPCASE	Converts lowercase characters to uppercase.

▪ Data Movement Statement

=	Moves the results of the expression on the right of the equal sign to the variable specified on the left of the equal sign.

Control Statements

Control statements govern the order of a program's instruction by modifying the normal sequence of statement execution. Some control statements call either internal or external subroutines (CALL, XCALL), some transfer control to other statements (GOTO), some execute a statement based on the results of a logical condition (IF). In addition, there are controls for disabling and enabling trapping of runtime errors (OFFERROR, ONERROR), for returning from subroutines (RETURN), for suspending program operation for specified intervals (SLEEP), and for terminating execution, and (optionally) chaining to another program (STOP).

Intertask Communications Statements

These statements allow communication between programs. The intertask communications statements are SEND and RECV.

Input/Output Statements

Input/output statements control the transmission and reception of data between memory and PDP-11 input/output devices such as the disk, the lineprinter, and the terminal. The input/output statements are

ACCEPT	Reads a character from a device.
CLOSE	Terminates use of an input/output channel and closes the associated file.
DELETE	Deletes a record from an ISAM file.
DISPLAY	Writes a character string to a device.
FORMS	Sends special form control characters used by lineprinters.
LPQUE	Requests printing of a file by a spooling program.
OPEN	Initializes a file in preparation for input/output operations.
READ	Reads a record from a file (direct access).
READS	Reads the next record in sequence from a file.
UNLOCK	Releases a file record for access by another program when operating in a timesharing environment.
WRITE	Writes a record to a file (direct access).
WRITES	Writes the next record in sequence to a file.

▪ Documentation Makes Transporting Your Applications Easy

Digital provides documentation for reference, development, and compatibility. These include

- *DIBOL for Beginners*

  This manual is an introduction to programming for DIBOL users with no previous programming experience.

- *Introduction to DIBOL-83*

  This manual includes an introduction to DIBOL and is intended for all Digital operating systems that support DIBOL-83.

- *The DIBOL-83 Language Reference Manual*

  This manual contains definitions and formats, along with rules of use and errors for each language statement.

- *The DIBOL User's Guide*

  This guide describes program development and has been produced for each Digital operating system on which DIBOL runs.

- *The DIBOL-83 Compatibility Guide*

  This guide was written for users who will be transporting applications across Digital operating systems. It explains the few remaining DIBOL differences and extensions, as well as describing the utilities used when moving programs and data files from system to system.

▪ You Can Debug Your Programs Quickly

The DIBOL Debugging Technique (DDT) allows you to interact with your DIBOL program while it is executing. Using DDT, you can set predetermined stopping points (called breakpoints) where you wish to temporarily suspend execution of the program. You can examine or alter the contents of variables using their symbolic names, trace through complicated sequences of subroutine nestings, and single step through lines of a DIBOL program. If you miss an error and the program aborts, DIBOL tells you the name of the program that failed and which statement caused the program to fail. These features allow you to locate problems, correct data values, and identify programming errors directly, before reediting, recompiling, and relinking your program.

[illegible]

[illegible]

- [illegible]

[illegible]

- [illegible]

[illegible]

- [illegible]

[illegible]

- [illegible]

[illegible]

- [illegible]

[illegible]

[illegible]

[illegible]

Chapter 16 ▪ PASCAL/RSX

▪ PASCAL/RSX Gives You Standard Pascal Features Plus Powerful Enhancements

Pascal is a high-level, structured programming language widely used in business, manufacturing, research, and education. Pascal's English-like commands, completely logical grammar, and block structure help developers produce programs that have clear organization and linear flow. The fact that programs are easily understood and maintained helps to control software maintenance costs.

PDP-11 PASCAL/RSX runs on all RSX-11M, Micro/RSX, or RSX-11M-PLUS based PDP-11 systems that have the Extended Instruction Set (EIS). It includes the features of the Level 0 ISO Specification for Computer Programming Language Pascal (DIS 7185) plus powerful enhancements designed to increase programmer efficiency.

▪ PDP-11 PASCAL/RSX Provides Powerful Extensions to Standard Pascal

Some PDP-11 PASCAL/RSX extensions enhance Pascal's modular capabilities and make it easier to develop portions of the same program independently.

These include

- The MODULE reserved word, which identifies separate modules for independent compilation.
- The %INCLUDE directive, which allows multiple compilation units to include the text of the same program.
- The EXTERNAL command, which allows references to subprograms and data external to the PASCAL program. This increases your ability to develop and compile modules separately.

Other extensions make PDP-11 PASCAL/RSX easier to use and help increase programmer productivity. These include

- The OTHERWISE clause in the CASE statement, which provides a succinct way to force execution of statements if the case selector does not match any of the case labels.
- The dollar sign ($) and underscore (_), which can be used in identifiers to enhance program readability.
- Value initialization in the declaration section of the program, which lets you assign initial values to variables, thereby helping to keep executable code size down.
- Predefined functions and procedures, which include OPEN, CLOSE, DATE, UPDATE, FIND, ADDRESS, TIME, and HALT.
- Language elements, which support sequentially organized File Control Services (FCS) files, giving you sequential access to files with variable-length records, and sequential and direct access to files with fixed-length records.
- STATIC, AUTOMATIC, and OVERLAID allocation attributes, which let you specify the type of storage that a variable or compilation unit should occupy.

PDP-11 PASCAL/RSX also contains extensions that enhance mathematical and computing capabilities. These are

- A REM operator to supply the remainder in division operations.
- Binary, hexadecimal, and octal constants.
- An exponentiation operator.

▪ You Have Convenient Access to the RSX Operating System

PDP-11 PASCAL/RSX programs can call RSX system services for process-control operations, execution of system directives, and special peripheral access.

PASCAL/RSX also provides a variety of compiletime and task building options and is able to support sequential File Control Services (FCS) files with fixed- or variable-length records.

Compiletime Options

PDP-11 PASCAL/RSX options can be specified in the compiler command line. These options include compiletime and runtime checks for array bounds, case selectors, pointers, string bounds, and subrange bounds. These options save valuable programming and debugging time by identifying many common errors.

Compiler options include optional machine code listings with line numbers to aid in debugging. To aid in program migration to other systems, you can call for listings with warning-level messages that identify the use of PASCAL/RSX extensions.

Applications using floating-point arithmetic can take advantage of the Extended Instruction Set (EIS) or Floating-Point Processor (FPP).

Task Builder Options

The Task Builder is used after compilation to produce an executable image file and to provide support for both resident and relocatable object libraries. Task Builder options create checkpointable tasks, identify the use of floating-point hardware, provide online debugging support through the Online Debugging Technique (ODT), and allow simultaneous execution of multiple versions of a single task.

Sequential or Direct Record Access
—Plus Fixed- or Variable-length Records

PDP-11 PASCAL/RSX supports sequential File Control Service (FCS) files that can have fixed-length or variable-length records. Even though FCS files are organized sequentially, PDP-11 PASCAL/RSX provides direct access on all files with fixed-length file organizations.

You select an access mode simply by specifying one parameter in the OPEN procedure. In direct access, the convenient FIND procedure lets you position the file pointer so that the GET or UPDATE acts on the exact file component you want. This easily coded, direct access can help you reduce execution time.

▪ Block Structuring Encourages Efficient Modular Programming

A Pascal program consists of two major components — a *HEADING* and a *BLOCK*. The HEADING defines the BLOCK's name and any external files the program will use for input and output. The BLOCK contains two sections of a program, procedure, or function — the *DECLARATION SECTION* and the *EXECUTABLE SECTION*.

Within the DECLARATION SECTION, all data and procedures to be executed must be specified. The EXECUTABLE SECTION contains the statements outlining actions to be performed within the BLOCK. BLOCKS can be nested within the procedure section of other BLOCKS.

The delimiters BEGIN and END are used to specify the beginning and end of the EXECUTABLE SECTION. They provide the framework for logical, yet flexible, programming techniques in conjunction with the following formatting capabilities:

- Free formatting of program text — You can place statements anywhere on a line, divide a statement across more than one line, or place several statements on one line.
- No line numbers — The semicolon (;) is the delimiter that separates successive Pascal statements. It is also used to terminate the program heading and the items in the declaration section.
- Comments can appear anywhere within a program and are denoted by brace enclosures ({}) or parentheses and asterisk enclosures ((**)).

A PDP-11 PASCAL/RSX compilation unit can be a program or a module, with the module consisting solely of a declaration section.

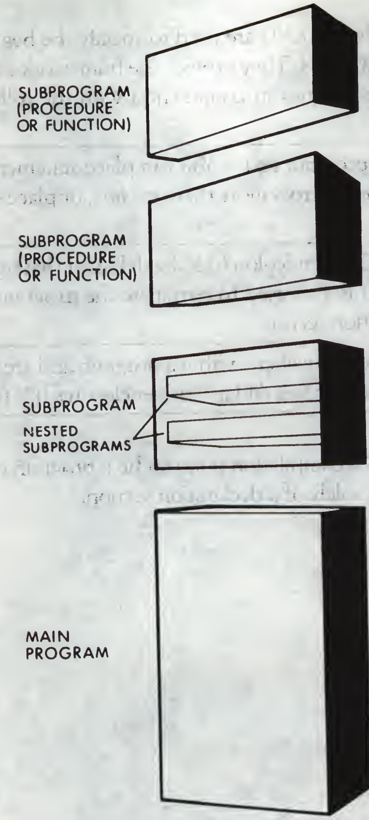


Figure 16-1 ▪ Block Structuring

▪ Each Constant, Variable, and Function Designator Is Implicitly or Explicitly Associated with a Data Type

The four categories of data types include ordinal types, the real type, structured types, and pointer types.

An *ordinal* type consists of an ordered sequence of values that can be represented by a sequence of successive integers; that is, any scalar type whose values are not real numbers.

PDP-11 PASCAL/RSX provides three predefined ordinal types (INTEGER, CHAR, and BOOLEAN) and two user-defined ordinal types (enumerated types and subrange types). These are described below.

Type INTEGER:	Consists of all the integers in the range −32,768 through 32,767. Can be written in binary, octal, decimal, and hexadecimal notations.
Type CHAR:	PDP-11 PASCAL/RSX uses an extended implementation of the ASCII character set. Can use the ORD function to convert any ASCII character to its corresponding ordinal value.
Type BOOLEAN:	Consists of the values FALSE and TRUE. ORD function returns the ordinal value of 0 for FALSE and 1 for TRUE.
Enumerated Types:	Consist of user-specified lists of constants. Defined by listing the values in order, separated by commas, within parentheses. ORD function can be used on the value of any enumerated type since each value is automatically associated within an ordinal number. Example: (Sun, Mon, Tue, Wed, Thu, Fri, Sat) is a valid enumerated type where the ordinal of Fri returns the value 5.
Subrange Types:	Consist of specified subsets of another data type. Specified by lo-bound (lower bound) and up-bound (upper bound). Example: The subrange type 6..12 consists of the integers 6,7,8,9,10,11,12.

REAL values can be expressed in either decimal or exponential notation. The range and precision of type REAL are

Smallest negative value:	−0.29E−38
Largest negative value:	−1.70E38
Smallest positive value:	0.29E−38
Largest positive value:	1.70E38
Precision:	1 part in 2**23 or 7 decimal digits

A *structured type* consists of component variables or constants that collectively define a Pascal record, array, set, or file. Structured data types can be processed either by aggregate (as a unit) or by individual components. They include

RECORD type:	A structured list of variables of the same or different data types.
ARRAY type:	A structured group of variables of the same data type that are ordered by means of one or more indexes (i.e., can be a multideminsional array).
SET type:	An unordered grouping of constants (up to 256) of the same ordinal type.
FILE type:	A sequence of logically related components (values) that in storage are arranged in physical order and treated as a unit.

A *pointer type* consists of the storage addresses of dynamic variables and the predeclared value NIL. A dynamic variable is an undeclared variable that is allocated by the system during program execution.

▪ Expressions Evaluate to a Single Value

An expression is any data item (constant, variable, or function designator) or any group of data items combined by operators that evaluate to a single value. Expression operators include

	Operator	Example	Result
Arithmetic	+	A+B	Sum of A and B
	–	A–B	Subtraction of B from A
	*	A*B	Product of A and B
	**	A**B	A raised to the power of B
	/	A/B	A divided by B
	DIV	A DIV B	Result of A divided by B, truncated toward zero
	REM	A REM B	Remainder of A divided by B
	MOD	A MOD B	Modulus of A with respect to B

	Operator	Example	Result
Relational operators	=	A=B	TRUE if A is equal to B
	<>	A<>B	TRUE if A is not equal to B
	>	A>B	TRUE if A is greater than B
	>=	A>=B	TRUE if A is greater than or equal to B
	<	A<B	TRUE if A is less than B
	<=	A<=B	TRUE if A is less than or equal to B
Logical operators	AND	A AND B	TRUE if both A and B are TRUE
	OR	A OR B	TRUE if either A or B is TRUE (or if both are TRUE)
	NOT	NOT A	TRUE if A is FALSE (and FALSE if A is TRUE)
String operators	=	A=B	TRUE if strings A and B have equal ASCII values
	<>	A<>B	TRUE if strings A and B have unequal ASCII values
	<	A<B	TRUE if ASCII value of string A is less than that of string B
	<=	A<=B	TRUE if ASCII value of string A is less than or equal to that of string B
	>	A>B	TRUE if ASCII value of string A is greater than that of string B
	>=	A>=B	TRUE if ASCII value of string A is greater than or equal to that of string B

	Operator	Example	Result
Set operators	+	A+B	Union of sets A and B
	*	A*B	Intersection of sets A and B
	–	A–B	Set of those elements of A that are not also in B
	=	A=B	TRUE if set A is equal to set B
	<>	A<>B	TRUE if set A is not equal to set B
	<=	A<=B	TRUE if set A is a subset of set B or is equal to set B
	>=	A>=B	TRUE if set B is a subset of set A or equal to set A
	IN	C IN B	TRUE if C is an element of B

▪ Statements Can Appear Anywhere in the Executable Section of a Program Block, Procedure Block, or Function Block

Simple statements have no component statements. The structured statements contain one or more component statements that can be simple statements or other structured statements. Statements include

Compound	Structured	Groups other Pascal statements into a single unit for consecutive execution.
Assignment	Simple	Assigns a value to a variable.
Empty	Simple	Any statement, partial statement, or absence of a statement that causes no action other than program execution to advance to the next statement.
CASE	Structured conditional	Causes execution of one or more component statements on the basis of the value of an ordinal expression.
IF-THEN	Structured conditional	Causes execution of a component statement if a relational or logical expression contained in the IF-THEN statement evaluates to TRUE.

IF-THEN-ELSE	Structured conditional	Causes execution of either of two components statements on the basis of whether a relational or logical expression evaluates to TRUE or FALSE.
FOR	Structured repetitive	Causes repetitive execution of a component statement on the basis of the value of an internally incremented or decremented variable.
REPEAT	Structured repetitive	Causes repetitive execution of one or more component statements until a Boolean expression evaluates to TRUE.
WHILE	Structured repetitive	Causes repetitive execution of a component statement for as long as a Boolean expression evaluates to TRUE.
WITH	Structured	Allows one to reference the fields of a record with an abbreviated notation.
GOTO	Simple	Causes an unconditional transfer of program control to a statement with a label.
Procedure call	Simple	Invokes a procedure.

▪ Routines Let You Break a Program Down into Component Parts

There are two kinds of routines in Pascal—procedures and functions. A procedure is a program unit that performs a certain operation or group of related operations. A function is a program unit that determines or computes a value and then returns this value to the block that called the function.

A PDP-11 PASCAL/RSX program can include user-written routines, predeclared routines, external routines, Object Time System routines, and routines written in MACRO-11. Routines can be included in a main program directly, by declaring them in a PROCEDURE or FUNCTION section, or can be used as separately compiled modules.

Predeclared routines supplied by PDP-11 PASCAL/RSX include

▪ Arithmetic functions:

ABS(x)	absolute value of x
ARCTAN(x)	arc tangent of x
COS(x)	cosine of x
EXP(x)	exponential of x
LN(x)	natural log of x
SIN(x)	sine of x
SQR(x)	square of x
SQRT(x)	square root of x

▪ Ordinal functions:

PRED(x)	Returns the value that immediately precedes x in the ordered sequence of values that compose the ordinal data type of which x is a member.
SUCC(x)	Returns the value that immediately succeeds x in the ordered sequence of values that compose the ordinal data type of which x is a member.

▪ Boolean functions:

ODD(x)	Tests whether x is odd or even and returns the value TRUE if x is odd, FALSE if x is even.
EOF(x)	Tests whether the file pointer is positioned beyond the end-of-file marker in file x and returns the value TRUE if it is, FALSE if it is not.
EOLN(x)	Tests whether the file pointer is positioned beyond the end-of-line marker in text file x and returns the value TRUE if it is, FALSE if it is not.

▪ Transfer functions:	
CHR(x)	Converts integer x in the range 0 through 255 to the character value in the ASCII character set whose ordinal value is x.
ORD(x)	Converts ordinal value x to the integer that represents the position of x in the ordered sequence of values that compose the data type of which x is a member.
ROUND(r)	Converts the REAL value r to an INTEGER value by rounding any fractional part of r.
TRUNC(r)	Converts REAL value r to an INTEGER value by truncating any fractional part of r.
▪ Transfer procedures:	
PACK	Converts an unpacked array into a packed array.
UNPACK	Converts a packed array into an unpacked array.
▪ Allocation routines:	
ADDRESS	Returns the address of either a static variable or a dynamic variable.
NEW	Allocates memory for a dynamic variable.
DISPOSE	Deallocates memory in heap storage for a dynamic variable created by the NEW procedure.
▪ Miscellaneous routines:	
DATE AND TIME	Assigns the current data and time, respectively, to a variable of type PACKED ARRAY [1..11] OF CHAR.
HALT	Causes program execution to stop.

▪ PASCAL/RSX Supports Both Sequential and Direct Access

A file in PDP-11 PASCAL/RSX is a sequence of logically related components (sometimes called records) that in storage are arranged in physical order and treated as a unit. PASCAL/RSX files are called sequential files. The components that make up any one file must all be of the same data type.

PASCAL/RSX supports two file access methods—sequential and direct. Under sequential access, the components of a file are processed in the sequence in which they are physically ordered. Under direct access, the components of a file are processed in any order and are selected for processing individually on the basis of their position relative to the physical beginning of a file. File components can be any data type, but all the components in any one file must be the same data type.

PASCAL/RSX supports two kinds of file components—fixed-length and variable length. Fixed-length components are file components that are of a specified size. Variable-length components may be of any size up to a specified maximum.

▪ PASCAL/RSX Provides Predeclared Routines for Performing I/O

The predeclared routines for performing I/O operations are

General procedures:

OPEN	Opens a file with specified characteristics.
CLOSE	Closes a file.

Sequential access input procedures:

GET	Reads a file component into the file buffer variable.
READ	Reads a file component into a specified variable.
RESET	Prepares a file for input.

Sequential access output procedures:

PUT	Writes the file buffer variable into the specified file.
REWRITE	Truncates a file to length zero and prepares it for receiving output.
WRITE	Writes specified values into a file.

Miscellaneous routine:

EOF	Indicates the end of an input file.

Text manipulation procedure:

EOLN	Indicates the end of an input file.
PAGE	Advances output to the next page of text file.
READLN	Reads a line from a text file.
WRITE	Allows one to specify field width to format the values being written in a text file.
WRITELN	Writes a line into a text file.

Direct access procedures:

FIND	Performs direct access to a file for input operations.
UPDATE	Performs direct access to a file for output operations.

▪ Attributes Allow Additional Control over the Properties of Variables, Routines, and Compilation Units

Table 16-1 lists the attributes and indicates the program elements with which each attribute can be associated.

Table 16-1 ▪ Summary of Attribute Use

Class	Attribute	Variable	Procedure of Function	Program or Module Heading
	GLOBAL	Yes	Yes	No
Visibility	EXTERNAL	Yes	Yes	No
	LOCAL	Yes	Yes	No
	STATIC	Yes	No	No
Allocation	AUTOMATIC	Yes	No	No
	OVERLAID	No	No	Yes

Figure 16-2 is a sample of the structure of a PDP-11 PASCAL/RSX program.

```
PROGRAM Calculator (INPUT, OUTPUT);

  VAR Subtotal, Operand : REAL;
      Equation : BOOLEAN;
      Operator: CHAR;
      Answer : CHAR;

PROCEDURE Instructions;
  BEGIN
    WRITELN ('This program adds, subtacts, multiplies, and');
    WRITELN ('divides real numbers. Enter a number in response');
    WRITELN ('to the Operand: prompt and enter an operator --');
    WRITELN ('+, -, *, /, or = in response to the Operator:');
    WRITELN ('prompt. The program keeps a running subtotal');
    WRITELN ('until you enter an equal sign (=) in response');
    WRITELN ('to the Operator: prompt. You can then exit from');
    WRITELN ('the program or begin a new set of calculations.');
  END; (* End of Procedure Instructions *)


BEGIN
WRITE ('Do you need instructions? Type Y or N. ');
READLN (Answer);
IF Answer='Y' THEN Instructions;
REPEAT
  Equation := FALSE;
  Subtotal := Ø;
  WRITE : ('Operand: ');
  READLN (Subtotal);

  WHILE (NOT Equation) DO
    BEGIN
      WRITE ('Operator: ');
      READLN (Operator);
      IF (Operator = '=') THEN
        BEGIN
          Equation := TRUE;
          WRITELN ('The answer is ',Subtotal);
        END (* End of If clause *)
      ELSE
        BEGIN
          WRITE ('Operand: ');
          READLN (Operand);
          CASE Operator OF
               '+' : Subtotal := Subtotal + Operand;
               '-' : Subtotal := Subtotal - Operand;
               '*' : Subtotal := Subtotal * Operand;
               '/' : Subtotal := Subtotal / Operand;
          END; (* End of CASE statement *)
          WRITELN ('The subtotal is ', Subtotal);
        END; (* End of ELSE clause *)
    END; (* End of WHILE statement *)

  WRITE ('Any more calculations? Type Y or N. ');
  READLN (Answer);
UNTIL Answer = 'N';
END.
```

Figure 16-2 ▪ Sample Structure of a PDP-11 PASCAL/RSX Program

Chapter 17 • File Management Utilities

PDP-11 operating systems provide a number of utilities and programs that are designed to make file management easy, both in the structuring and in the accessing of files. While the descriptions that follow do not exhaust the list of Digital's file management programs or utilities, they do suggest the breadth of products available.

▪ File Control Services (FCS) Provide a User Interface to the File System

RSX-11 File Control Services enable you to perform record-oriented and block-oriented I/O operations and to perform additional functions required for file control, such as open, close, wait, and delete operations. To invoke FCS functions, the user issues macro calls to specify desired file control operations. The FCS macros are then called at assembly time to generate code for specified functions and operations. Figure 17-1 illustrates the file access operation.

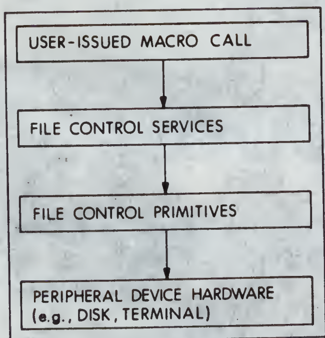


Figure 17-1 ▪ File Access Operation

FCS is a set of routines linked with the user program at taskbuild time from a resident system library or a system object module library. These routines, consisting of pure, position-independent code, provide a user interface to the file system, enabling the user to read and write files on file-structured devices and to process files in terms of logical records.

FCS allows the user to write a collection of data (consisting of distinct logical records) to a file in a way that enables retrieving the data at will. Data can be retrieved from the file without having to know the exact form in which it was written to the file. FCS thus provides a sense of transparency to the user so that records can be read or written in logical units that are consistent with an applications requirement.

File Access Method

Under FCS, RSX-11 supports both sequential and direct access to files. The sequential access method is device-independent. That is, it can be used for both record-oriented and file-structured devices (for example, card reader and disk, respectively). The direct access method can be used only for file-structured devices.

Data Formats for File-structured Devices

Data is transferred between peripheral devices and memory in blocks. A data file consists of virtual blocks. Each virtual block contains one or more logical records. Records in a virtual block can be either fixed or variable in length.

Virtual blocks and logical records within a file are numbered sequentially, starting with one. A virtual block number is a file-relative value, while a physical block number is a volume-relative value. For example, the first virtual block in a file is always virtual block number 1, but at the same time it could also be physical block number 156.

Block I/O Operations

The READ and WRITE macro calls allow the user to read and write virtual blocks of data from and to a file without regard to logical records in a file. Block I/O operations provide a very efficient means of processing file data, because such operations do not involve the blocking and deblocking of records within the file. Also, in block I/O operations, the user can read or write files in an asynchronous manner; control can be returned to the user program before the request I/O operation is completed.

When block I/O is used, the number of the virtual block to be processed is specified as a parameter in the appropriate READ and WRITE macro call. The specified virtual block is processed directly in a buffer reserved by the program in its own memory space.

The user is responsible for synchronizing all block I/O operations. Such asynchronous operations can be coordinated through an event flag specified in the READ and WRITE call. The event flag is used by the system to signal the completion of the I/O transfer, enabling the user to coordinate those block I/O operations that depend on each other.

Record I/O Operations

The GET and PUT macro calls are provided for processing record-oriented files. GET and PUT operations perform the necessary blocking and deblocking of the records within the virtual blocks of the file, allowing the user to read or write individual records.

In preparing for record I/O operations, the user program must specify the format of the records. For example, it must specify whether the records are fixed or variable in length, or whether records that are to be output to a carriage-control device are to contain carriage-control information, which can be either at the beginning of the record or embedded within the records.

For sequential access files, I/O operations can be performed for both fixed and variable length records. For direct access files, I/O operations can be performed only for fixed length records.

In contrast to block I/O operations, all record I/O operations are synchronous; control is returned to the user program only after the requested I/O operation is performed.

Because GET and PUT operations process logical records within a virtual block, only a limited number of GET or PUT operations result in an actual I/O transfer, that is, when the end of a data block is encountered. Therefore, not all GET and PUT I/O requests will necessarily involve a physical transfer of data.

The File Storage Region

The file storage region (FSR) is an area allocated in the user program as the working storage area for record I/O operations. The FSR consists of two program sections that are always contiguous to each other. The first program section of the FSR contains the block buffers and the block buffer headers for record I/O processing. The user determines the size of the area at assembly time. The number of block buffers and associated headers is based on the number of files that the user intends to open simultaneously for record I/O operations.

The second program section of the FSR contains impure data that is used and maintained by FCS in performing record I/O operations. Portions of this area are initialized at taskbuild time, and other portions are maintained by FCS. This program section is intentionally isolated from the user to preserve its integrity.

Blocking and deblocking of records during input is accomplished in the FSR block buffer during output. Note also that the FCS serves as the user interface to the FSR block buffer pool. All record I/O operations initiated through GET and PUT calls are totally synchronized by FCS.

Data Transfer Modes

When record I/O is used, a program can gain access to a record in either of two ways after the virtual block has been transferred into the FSR from a file:

- Move mode—Individual records are moved from the FSR buffer. Move mode simulates the reading of a record directly into a user record buffer, thereby making the blocking and deblocking of records transparent to the user.
- Locate mode—The user program accesses records directly in the FSR block buffer. Program overhead is reduced in locate mode, because records can be processed directly within the FSR block buffer.

Shared Access to Files

FCS permits shared access to files. Several MACRO calls are available in FCS for opening files. Two of these calls are OPNS and OPEN. The OPNS macro call is used specifically to open a file for shared access. The OPEN call invokes generalized open functions that have shared access.

OPNS allows several active read-access requests and one write-access request for the same file. OPEN allows multiple read-access requests for the same file but does not permit concurrent write access. Note that shared access during reading does not necessarily imply the presence of read requests from several separate tasks. The same task can open the same file using different logical unit numbers.

Spooling Operations

FCS provides facilities at both the macro and subroutine level to queue files for subsequent printing. A task issues the PRINT macro call to queue a file for printing on the system lineprinter.

FCS Macros and Macro Use

FCS includes four basic kinds of macros that simplify the user's interface to the system's file control primitives. The four kinds are

- Initialization macros.
- File processing macros.
- Command line processing macros.
- The CALL macro.

The initialization and file processing macros are used to establish the database description and the necessary temporary storage areas needed to perform I/O operations. The command line processing macros are used to process I/O commands entered from a terminal. The CALL macro is used to invoke file control routines.

The initialization and file processing macros set up the following structures to define the database:

- A file descriptor block (FDB) that contains execution-time information necessary for file processing. It defines the basic characteristics of a file, e.g., record type, record size, and access privileges.
- A data set descriptor that is accessed by FCS to obtain the file name, type, version number, and location necessary to open a specified file. The data set descriptor is used when a program accesses a given set of known or predefined files.
- A default file name block that is accessed by FCS to obtain default file information required to open a file. This is accessed when complete file information is not specified in the data set descriptor. It is used by programs written to access a general set of files.

There are two types of initialization macros — assembly time macros and runtime macros. Data supplied during assembly of the source program establishes the inital values in the FDB. Data supplied at runtime can either initialize additional portions of the FDB or change values established at assembly time. Furthermore, the data supplied through the file-processing macros can either initialize portions of the FDB or change previously initialized values. The user not only has a broad range of control over defining the database characteristics but also has control over when the definitions are made.

File processing macros also determine the way in which files are processed. These macro calls are invoked and expanded at assembly time. The resulting code is then executed at runtime to perform the following operations:

OPEN	Opens and prepares a file for processing.
OPNS	Opens and prepares a file for processing; allows shared access to the file (depending on the mode of access).
OPNT	Creates and opens a temporary file for processing.
OFID	Opens an existing file using the file identification provided in the filename block.
GET	Reads logical records from a file.
GETR	Reads fixed-length records from a file in random access mode.
GETS	Reads records from a file in sequential access mode.
PUT	Writes logical records to a file.
PUTR	Writes fixed-length records to a file in random mode.
PUTS	Writes records to a file in sequential mode.

READ Reads virtual blocks from a file.

WRITE Writes virtual blocks to a file.

DELETE Removes a specified file from the associated volume directory and deallocates the space occupied by the file.

WAIT Suspends program execution until a requested block I/O is performed.

PRINT Queues a file for printing on a special terminal or lineprinter.

In summary, the file-processing macros allow the user to specify random access or sequential access to files and perform block-oriented or record-oriented file processing. In addition, the PRINT macro allows the user to spool files to a lineprinter or terminal device.

The command line processing macros allow the user to access special routines available in the system object library. The Get Command Line (GCML) routine accomplishes all the logical functions associated with the entry of a command line from a terminal, an indirect command file, or an online storage medium. The Command String Interpreter (CSI) routine takes command lines from the GCML input buffer and parses them into appropriate data set descriptors required by FCS for opening files.

The CALL macro allows the user to access a special set of file control routines. These routines allow a MACRO program to perform the following operations—find, insert, or delete a directory entry, rename a file, extend a file, mark a temporary file for deletion, and delete a file, among other operations.

▪ SORT Runs on Operating Systems with RMS

The SORT utility program allows you to reorder data from any input file into a new file in either ascending or descending sequence based upon control or key fields within the input data records themselves. SORT runs under any operating system that includes RMS (Record Management Services).

If you do not wish to sort the actual data, SORT can still be used to extract key information. Then the user can sort that information and store the information on a permanent file. Later that file can be used to access the data in the order of the key information on the sorted file. The contents of the sorted file may be entire records, key fields, or record indexes relative to the position of each record within the file (the first record on the database is record 1, the second, 2, etc.). SORT provides four sorting techniques that are outlined later in this chapter.

The SORT utility program may be controlled by a command string and an optional specification file. There is a simple format for each. If your SORT application does not require that records be restructured or that only a subset of the input file be sorted, then only a command string is needed to control SORT.

Data Files

SORT can accept a file from any one of the peripheral devices available in the system configuration: disk units, magtape units, or terminals.

A record is usually divided into several logical areas called data fields. The data in each field may or may not be relevant to SORT. SORT uses record identifiers to distinguish the various types of records in a file while it uses the key fields in each record to reorder an input file. The key fields may be any one of a number of different data types including character, zoned decimal, two's-complement binary, and 2- or 4-word floating point. Any other data field in a record may be retained in the output file or ignored.

Table 17-1 ▪ Sorting Processes and Devices That Best Suit Specific Processing Environments

Sorting Technique	Input File	Output File	Work File
SORTR (Record Sort)	Disk	Disk	Disk
	Magtape*	Magtape*	3-8 files
	Paper Tape	Paper Tape	
	Cards	Printer	
	Console	Console	
SORTT (Tag Sort)	Disk	Disk	Disk
		Magtape*	3-8 files
		Console	
		Paper Tape	
SORTA (Address routing Sort)	Disk	Disk	Disk
			3-8 files
SORTI (Index Sort)	Disk	Disk	Disk
			3-8 files

*Provided records are at least 18 bytes long. Magtape must be in ANSI format.

Command String and Specification File

The user can direct the SORT program by entering a command string, which serves three functions. These functions will

- Reference devices in the system for each file in the current sort.
- Specify switches that define file parameters used in the sorting process.
- Reference a specification file or specify other switches to control the sort.

Several command string switches define the sorting process parameters. One switch describes record formats and the maximum record size. Another delimits the internal work files. Others provide detailed file information to RMS.

Normally, the sort must be directed with a specification file, but two additional switches may be used instead of a specification file. The first specifies the sorting process option; the second identifies the key fields. The use of these switches is limited to sorting an input file of uniform format:

- The key fields must reside in the same location in every record of the input file.
- The file must contain only the records to be included in the sort.

Figure 17-2 illustrates a general sort that would require only a command string and switches.

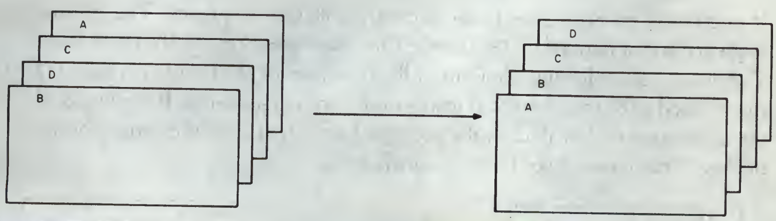


Figure 17-2 ▪ Sort Using Command String and Switches

The specification file is the supplement to the command string that provides the basis for controlling and directing the sorting process.

The specification file provides a variety of controlling features. They are record selection, alternate collating sequence, forced key, input format variation, and output format variation (Figures 17-3–17-7).

Figure 17-3 ▪ Record Selection

You can include or omit any records from the sorting process. The output file will contain only the specified records.

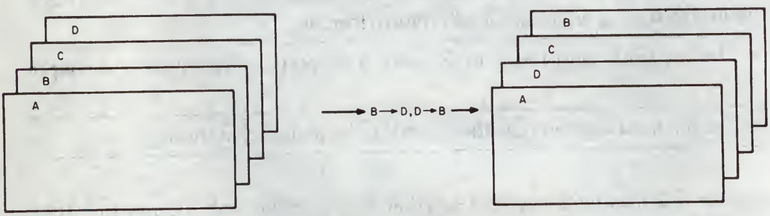


Figure 17-4 ▪ Alternate Collating Sequence

If necessary, you can specify an alternate collating sequence. The normal sequence is that implied in ASCII code. One alternate choice is EBCDIC values. The other is an individual alternate collating sequence (ALTSEQ). An ALTSEQ can be used to change the ASCII values of the normal sequence. It applies to all the alphanumeric key data in the records, but only during the actual sorting process. The output record remains unchanged.

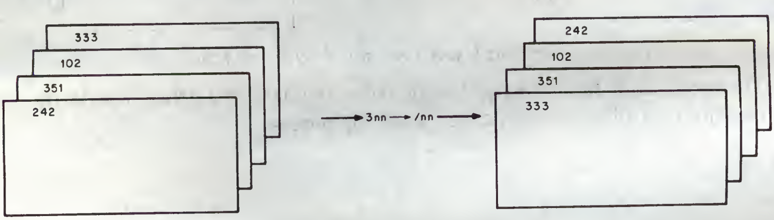


Figure 17-5 ▪ Forced Key

An ALTSEQ applies to all positions of the key. Forced keys allow the user to specify an alternate sequence for particular positions within the key. An alternate can be specified by substituting a lower-valued character, such as the slash (/). Because the slash comes before 0, the 300-series records in the example are brought to the front of the file. Note that the records so treated are in sequence and in front of the rest of the sorted file. The net effect is that of two sorted files, one behind the other.

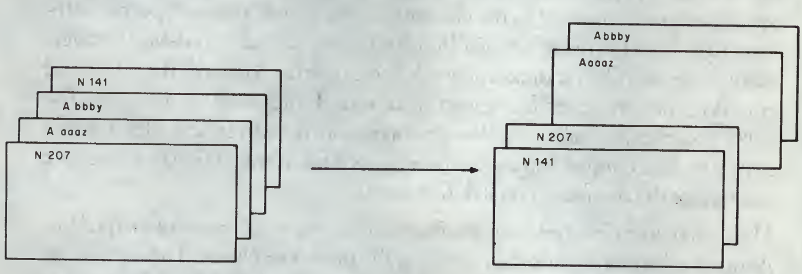


Figure 17-6 ▪ Input Format Variation

If the input file contains records with several different formats, the user can identify those records by type so that they may be properly handled. Note that only one type may be selected and sorted per run. In the example above, A and N are record identifiers.

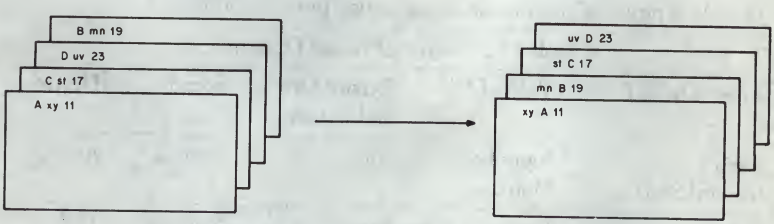


Figure 17-7 ▪ Output Format Variation

You can change the format of the data file during the sort, but you cannot change the contents of any given data item.

SORT Operation

The SORT program consists of two basic parts—a control program and a subroutine package called SORTS. The control program directs the overall processing, while SORTS serves as a collection of subroutines available to the control program during its processing. The subroutine package can be invoked from a user-written program. This is supported in most PDP-11 programming languages.

In the SORT control program are three phases of operation. In the first phase, SORT reads the command string, decodes it, and stores the switch values and the specification file, if present. Any errors in the command string or specification file are reported at this point.

Phase two begins the presort operation. The control program is called to open and read the input file and establish the keys. The SORTS subroutine begins the initial sorting process. At this point, the amount of available internal storage space becomes important to the efficiency of the sort. If that space is not sufficient to hold all the records, SORT builds strings of sorted records and transfers them to scratch files on bulk storage devices. In order to merge these files and complete the sort, space for at least three scratch files must be available. The SORT program normally provides for a maximum of eight scratch files. Either a switch in the command string or the amount of available internal work space can reduce the number of scratch files used.

The final merge phase rebuilds the intermediate scratch files into a merged file. Another subroutine reads the records in the proper sequence. The records are then written into the output file. If there are no scratch files to merge because main memory was sufficient to hold all the records, the sorted records are written directly into the output file. After the last record is written, the control program cleans up the scratch files and returns to the first phase; SORT is then ready to accept another job.

Table 17-2 provides information about sorting process options.

Table 17-2 ▪ Sorting Process Options

Type of SORT	Type of File	Record Size and Format	Speed	Device
SORTR (Record Sort)	Input and Output	Any	Slowest	Any
SORTT (Tag Sort)	Input	Any	Slow for large file	Disk
	Output	Any		Any
SORTA	Input	Any	Fastest	Disk
(Address Routing SORT)	Output	Fixed, six bytes		Any
SORTI (Index Sort)	Input	Any	Fast	Disk
	Output	Fixed, 6-byte pointer + original key		Any

Record Sort (SORTR)

The Record Sort (SORTR) outputs all specified record data in a sorted sequence. Each record is kept intact throughout the entire sorting process. Because it moves the whole record, SORTR is relatively slow and may require considerable main memory or external storage work space for large files.

Tag Sort (SORTT)

The Tag Sort (SORTT) produces the same kind of output file as SORTR, but it handles only record pointers and key fields. Because SORTT moves a smaller amount of data than SORTR, SORTT usually performs a faster sort than SORTR. The input file must be randomly reaccessed to create the entire output file, which may be a lengthy process for large files.

Address Routing Sort (SORTA)

SORTA produces address routing files, which consist of relative record pointers, beginning at 1, in binary words. These files can be used as a special index file to randomly access the data in the original file. It is possible to maintain only one data file, but several different index files are needed. Like SORTT, SORTA uses the minimum amount of data necessary in the sorting process. Once the input phase is completed, the input file is not read again. The output data is in a restricted mode. This means that SORTA is the fastest sorting method in the sort package.

Index Sort (SORTI)

SORTI produces an index file consisting of relative record pointers, as in SORTA, and index keys. This makes it slightly slower than SORTA. During processing, SORTI handles only the relative record pointers and two forms of the key fields. One form is used for sorting and the other is left as it was in the original data.

Other Utilities Help Make the Operations You Need Easier and Quicker

Running across most systems is a group of file management utilities that help make the various operations you need easier and quicker. The major features of these utilities are listed below.

PIP — The Peripheral Interchange Program (PIP) is a general purpose file utility package for the general user, programmer, and the system manager. PIP normally handles all files with the operating systems standard data formats. In general, the program transfers data files from any device in the system to any other device in the system. PIP can also delete or rename any existing file. Some operating systems include special file management operations in the PIP utility, such as directory listings, device initialization and formatting, and account creation.

FILEX — A File Exchange program, FILEX is a special purpose file transfer utility similar in operation to PIP. It provides the ability to copy files stored in one kind of format to another format. This enables a user to create data on one system in a special format and then transfer the data to a device in a format that another system can read. (Called FLX on RSX-11 systems.)

DUMP — DUMP displays all or selected portions of a file on a terminal or lineprinter. In general, DUMP enables the user to inspect the file in any of three modes — ASCII, byte, and octal. In ASCII mode, the content of each byte is printed as an ASCII character. In byte mode, the content of each byte is printed as an octal value. In octal mode, the content of each word is printed as an octal value. (DMP for RSX-11 systems.)

VERIFY — In general, a VERIFY program checks the readability and validity of data on a file-structured device. (VFY, for RSX-11 systems.)

DUP — The device utility program (DUP) is a device maintenance utility program. DUP creates files on file-structured RT-11 devices only. It can also extend files on certain file-structured devices such as disks, and it can compress, image copy, initialize, or boot RT-11 file structured devices. DUP does not operate on nonfile-structured devices such as lineprinters or terminals.

DSC — DSC enables the user to backup/restore disk volumes to magnetic tape or other disks and to combine unused blocks on disks to create contiguous blocks on RT-11 systems only. DSC comes both as a stand-alone and an online program.

CMP — CMP is a utility that will compare, line by line, two ASCII files. Its output can be either a new file with all the differences encountered, a listing of one with change bars marking the differences, or an output suitable for input to the SLP utility.

BRU — BRU backs up on disk to a tape in less than an hour, which is approximately a 4-to-1 improvement over the current DSC program. BRU also supports incremental backups (such as backing up only the files that have been modified since the previous backup), greatly reducing the amount of time required for proper disk backup.

Chapter 18 • Record Management Services (RMS)

▪ RMS Organizes and Accesses Records within Files

Record Management Services, a set of general purpose file-handling capabilities, combines with a host operating system to provide efficient and flexible data storage and modification on the host or remote operating system. When writing programs, you can select processing methods suitable to your application from among several RMS file structuring and accessing techniques.

RMS handles such functions as file organization and access methods and also manages the other file attributes (for example, storage medium and record format) and the runtime environment. By accomplishing most of its work transparently, RMS relieves programmers of many of the complexities associated with file and record manipulation.

RMS on the PDP-11 family is called RMS-11. It's included as part of RSX-11M-PLUS, RSX-11M, and RSTS/E and P/OS, the operating system of the Professional 300 series of personal computers. The RMS-11 utilities are included in VAX-11/RSX on the VMS operating system. RMS also provides a common data format for transport among these systems. Through RMS, programmers can maintain data files created by BASIC-PLUS-2, COBOL-81, MACRO-11, and DIBOL programming languages. Depending on the language in which programs are written, various RMS capabilities are available. When writing programs, programmers can select processing methods among the RMS file organizations and accessing techniques.

▪ Files Collect Related Information

A *file* is a collection of related information whose requirements are established by the nature of application programs needing the information. A company might maintain personnel information (employee names, addresses, job titles) in one file, for example, and product information (part numbers, prices, specifications) in a second, separate file. Within each of these files, the information is divided into *records*. In the personnel file, it would be logical for all the information on a single employee to constitute a single record and for the number of records in the file to equal the number of employees. Similarly, each record in the product information file would represent a description of a single product. The number of records in the file reflects the requirements of a particular application, in this case, a central registry of products sold by a company.

Each record in the personnel and product files would be subdivided into discrete pieces of information known as *data fields*, whose number, location within the record, and logical interpretation are defined by the programmer. Program applications then interpret, for instance, a particular data field in records of the personnel file as the name of an employee. They would interpret another data field in records of the product file as a part number. Figure 18-1 illustrates records that might occur in a personnel file and in a product file.

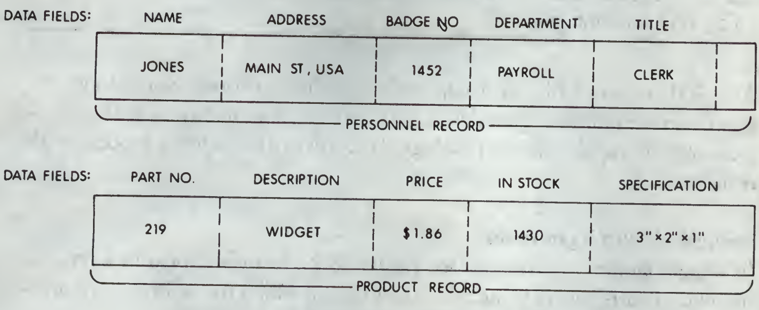


Figure 18-1 ▪ Personnel and Product Records

Thus, the relationships among data fields and records are known and are embedded in the logic of the programs. RMS does not require an awareness of such logical relationships; rather, RMS processes records as single units of data. *Programs either build records and pass them to RMS for storage in a file or issue requests for records while RMS performs the necessary operations to retrieve the records from a file.*

The purpose of RMS, then, is to ensure that every record written into a file can subsequently be retrieved and passed to a requesting program as a single logical unit of data. The structure, or *organization*, of a file establishes the manner in which RMS stores and retrieves records. The way a program requests the storage or retrieval of records is known as the *access mode*. Legal access modes depend on the organization of a file.

▪ Three File Organizations Govern the Relationships of Records

When creating a file, you have a choice of three file organizations:

- Sequential.
- Relative.
- Indexed.

RMS also supports three record access modes (techniques for storing and retrieving file records): sequential, random, and Record's File Address (RFA).

Among the attributes specified when creating an RMS file are

- Storage medium.
- Filename and protection specifications.
- Record format and size.
- File allocation information.

After RMS creates a file according to the specified attributes, application programs can use RMS access modes to store, retrieve, and modify data. These program operations take place on the logical records in a file or in the blocks the file comprises.

Sequential File Organization

In sequential file organization (see Figure 18-2), records appear in a physical sequence that is always identical to the order in which the records were originally written to the file by an application program.

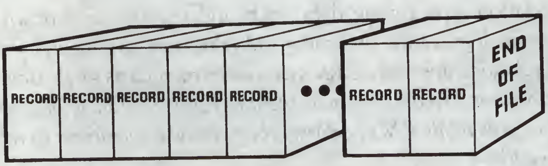


Figure 18-2 ▪ Sequential File Organization

Relative File Organization

When relative organization is selected, RMS structures a file as a series of fixed-size record cells. Cell size is based on the size specified as the maximum permitted length for a record in the file. RMS considers these cells as successively numbered from 1 (the first) to *n* (the last), and the cell's number represents its location *relative* to the beginning of the file.

Each cell in a relative file can contain a single record. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells containing records.

Because cell numbers in a relative file are unique, they can be used to identify both a cell and the record (if any) occupying that cell. Thus, record number 1 occupies the first cell in the file, record number 17 occupies the seventeenth cell, and so forth. When a cell number is used to identify a record, it is also known as a *relative record number*. Figure 18-3 depicts the structure of a relatively organized file.

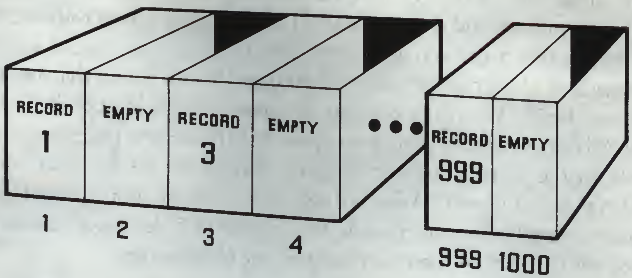


Figure 18-3 ▪ Relative File Organization

Indexed File Organization

Unlike the physical ordering of records in a sequential file or the relative positioning of records in a relative file, the location of records in indexed file organization is transparent to the program. RMS completely controls the placement of records in an indexed file. The presence of *keys* in the records of the file governs this placement.

The key chosen by the programmer, is a data type present in every record of a particular indexed file. The location and length of this data type are attributes of the file, and therefore are identical for all records in the given file. Legal key data types are string (1-255 bytes), integer (2 and 4 bytes), unsigned binary (2 and 4 bytes), and packed decimal (1-16 bytes). Selecting a data type indicates to RMS that the content (that is, key value) of that key in any particular record written to the file can be used by a program to identify that record for subsequent retrieval. Because the key is the arbitrary choice of the programmer, it can, of course, be equal to a field. Therefore, in the inventory file, the part number field could be a key; in the personnel file, the last name field could be a key.

At least one key, the *primary key*, must be defined for every indexed file. Optionally, additional *alternate keys* can be defined. Each alternate key represents an additional character string in records of the file. The key value in any one of these additional strings can also be used to identify the record for retrieval.

As programs write records into an indexed file, RMS locates the values contained in the primary and alternate keys. From the values in keys within records, RMS builds a tree-structured table known as an index, consisting of a series of entries each of which contains a key value copied from a record that a program wrote into the file. With each key value is a pointer to the location in the file of the record from which the value was copied. RMS builds and maintains separate indexes for the primary and alternate keys defined for the file. Each index is stored in the file. Figure 18-4 shows the general structure of an indexed file that has been defined with only a single key. Figure 18-5 depicts an indexed file defined with two keys—a primary key and one alternate key.

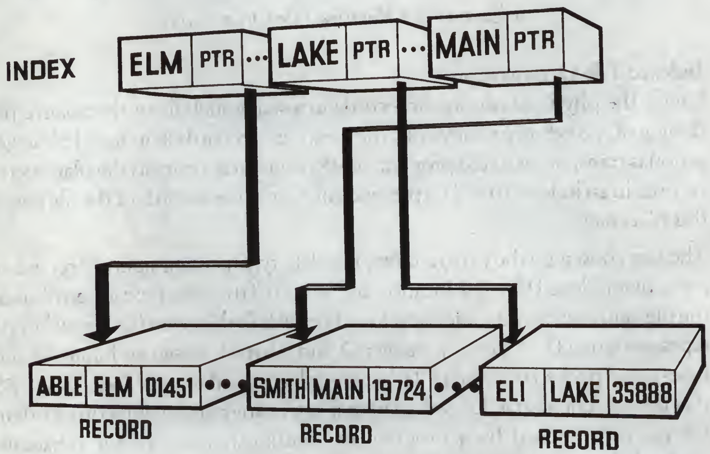


Figure 18-4 ▪ Single-key Indexed File Organization

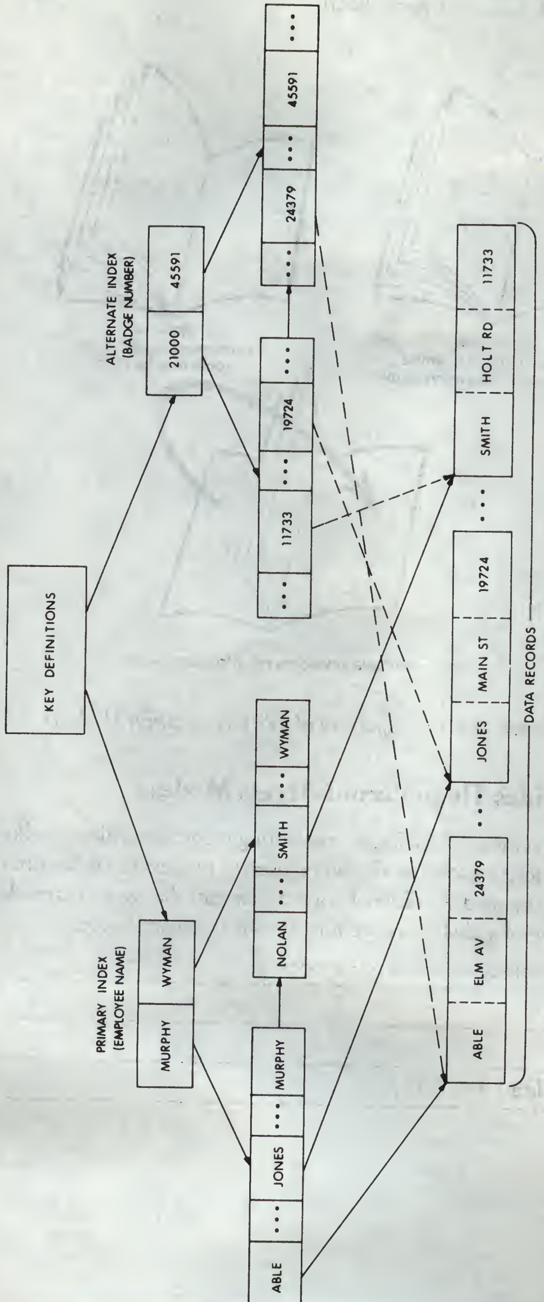


Figure 18-5 ▪ Multikey Indexed File Organization

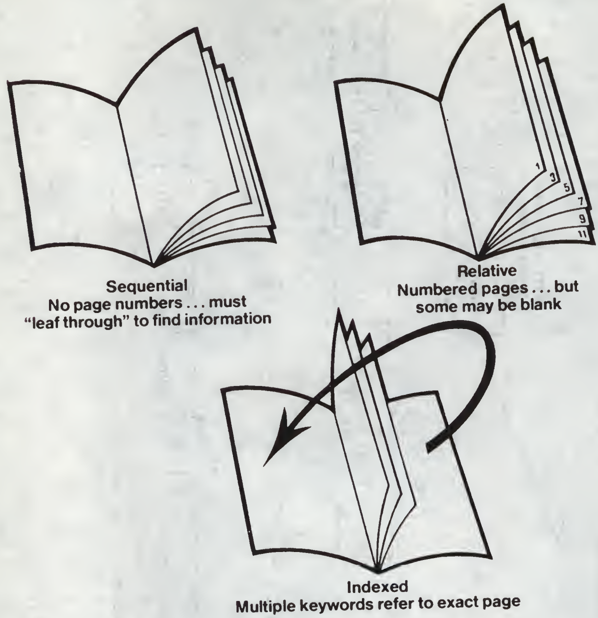


Figure 18-6 ▪ Comparison of File Organization Methods

▪ RMS Provides Three Record-Access Modes

The various methods of retrieving and storing records in a file are called access modes. A different access mode can be used to process records within the file each time it is opened. Additionally, a program can change access mode during the processing of a file by a procedure known as dynamic access.

RMS provides three record-access modes:

- Sequential.
- Random.
- Record's file address (RFA).

For logical reasons, RMS permits only certain combinations of file organization and access mode. Table 18-1 lists these combinations.

Table 18-1 ▪ Permissible Combinations of Access Modes and File Organizations

File Organization	Access Mode			
	Sequential	Random		RFA
		Record #	Key Value	
Sequential	Yes	No	No	Yes*
Relative	Yes	Yes	No	Yes
Indexed	Yes	No	Yes	Yes

*Disk files only

Sequential Access Mode

Sequential access mode can be used with any RMS file. Sequential access means that records are retrieved or written in a particular sequence. The organization of the file establishes this sequence.

- SEQUENTIAL ACCESS TO SEQUENTIAL FILES

In a sequentially organized file, physical arrangement establishes the order in which records are retrieved when using sequential access mode. To read a particular record in a file, say the fifteenth record, a program must open the file and access the first fourteen records before accessing the desired record. Thus, each record in a sequential file can be retrieved only by first accessing all records that physically precede it. Similarly, once a program has retrieved the fifteenth record, it can read all the remaining records (from the sixteenth on) in physical sequence. It cannot, however, read any preceding record without beginning again with the first record.

When writing new records to a sequential file in sequential access mode, a program must first request that RMS position the file immediately following the last record. Then each sequential write operation the program issues causes a record to be written following the previous record.

- SEQUENTIAL ACCESS TO RELATIVE FILES

During the sequential access of records in the relative file organization, the contents of the record cells in the file establish the order in which a program processes records. RMS recognizes whether successively numbered record cells are empty or contain records.

When a program issues read requests in sequential access mode for a relative file, RMS ignores empty record cells and searches successive cells for the first one containing a record. If, for example, a relative file contains records only in cells 3, 13, and 47, successive sequential read requests cause RMS to return relative record number 3, then relative record number 13, and finally relative record number 47.

When a program adds new records in sequential access mode to a relative file, the order in which RMS writes the records depends on ascending relative cell numbers. Each write request causes RMS to place a record in the cell whose relative number is one higher than the relative number of the previous request, as long as that cell does not already contain a record. If the cell already contains a record, RMS rejects the write operation. Thus, RMS allows a program to write new records only into empty cells in the file.

▪ SEQUENTIAL ACCESS TO INDEXED FILES

In an indexed file, the presence of one or more indexes permits RMS to determine the order in which to process records in sequential access mode. The entries in an index are arranged in ascending order by key values. Thus, an index represents a logical (rather than physical) ordering of the records in the file. If more than one key is defined for the file, each separate index associated with a key represents a different logical ordering of the records in the file. A program, then, can use the sequential access mode to retrieve records in the order represented by any index.

When reading records in sequential access mode from an indexed file, a program initially specifies a key (primary key, first alternate key, second alternate key, and so on) to RMS. Thereafter, RMS uses the index associated with that specified key to retrieve records in the sequence represented by the entries in the index. Each successive record RMS returns in response to a programmed read request contains a value in the specified key field that is equal to or greater than that of the previous record returned.

In contrast to a sequential read request, sequential write requests to an indexed file do not require the initial key specification. Rather, RMS uses the stored definition of the primary key field to locate the primary key value in each record to be written to the file. When a program issues a series of sequential write requests, RMS verifies that each successive record contains a key value in the primary key field that is equal to or greater than that of the preceding record.

Random Access Mode

In random access mode, the program, rather than the organization of the file, establishes the order in which records are processed. Each program request for access to a record operates independently of the previous record accessed. Associated with each request in random mode is an identification of the particular record of interest. Successive requests in random mode can identify and access records anywhere in the file.

Random access mode cannot be used with sequentially organized files. Both the relative and indexed file organizations, however, permit random access to records. The subsections that follow describe the use of random access with these organizations. Each organization provides a distinct way programs can identify records for access.

- RANDOM ACCESS TO RELATIVE FILES

Programs can read or write records in a relative file by specifying relative record number. RMS interprets each number as the corresponding cell in the file. A program can read records at random by successively requesting, for example, record number 47, record number 11 and, record number 31. If no record exists in a specified cell, RMS returns a nonexistence indicator to the requesting program. Similarly, a program can store records in a relative file by identifying the cell in the file that a record is to occupy. If a program attempts to write a new record in a cell already containing a record, RMS returns a record-already-exists indicator to the program.

- RANDOM ACCESS TO INDEXED FILES

The indexed file organization also permits random access of records. However, for indexed files, a key value rather than a relative record number identifies the record.

Each program read request in random access mode specifies a key value and the index (primary index, first alternate index, second alternate index, and so on) that RMS must search. When RMS finds the key value in the specified index, it reads the record that the index entry points to and passes the record to the user program. Under random access the programmer could, for example, instruct RMS to return all records with SMITH in the key-equal-to-last-name field.

In contrast to read requests, which require a program-specified key value, program requests to write records randomly in an indexed file do not require the separate specification of a key value. All key values (primary and, if any, alternate key values) are in the record itself. When an indexed file is opened, RMS retrieves all definitions stored in the file. Thus, RMS knows the location and length of each key field in a record. Before writing a record into the file, RMS examines the values contained in the key fields and creates new entries in the indexes. In this way RMS ensures that the record can be retrieved by any of its key values. Thus, the process by which RMS adds new records to the file is precisely the process it uses to construct the original index or indexes.

Record's File Address (RFA) Access Mode

Record's file address (RFA) access mode can be used with any file organization as long as the file resides on a disk device. This access mode is further limited to retrieval operations only. Like random access mode, however, RFA access allows a specific record to be identified for retrieval.

As the name suggests, every record within a file has a unique address. The actual format of this address depends on the organization of the file. In all instances, however, only RMS can interpret this format.

The most important feature of RFA access is that the address (RFA) of any record remains constant while the record exists in the file. After every successful read or write operation, RMS returns the RFA of the subject record to the program. The program can then save this RFA to use again to retrieve the same record. It is not required that this RFA be used only during the current execution of the program. RFAs can be saved and used at any later time.

Dynamic Access

Dynamic access is not strictly an access mode. Rather, it is the capability to switch from one access mode to another while processing a file. There is no limitation on the number of times such switching can occur. The only limitation is that the file organization (or, in the case of RFA access, the device containing the file) must support the access mode selected.

As an example, dynamic access can be used effectively immediately following a random or RFA access mode operation. When a program accesses a record in one of these modes, RMS establishes a new current position in the file. Programs can then switch to sequential access mode. By using the randomly accessed record (rather than the beginning of the file) as the starting point, programs can retrieve succeeding records in the sequence established by the file's organization.

▪ File Attributes Are Defined at Creation Time

The logical and physical characteristics of an RMS file are known as its attributes. These characteristics are defined by the source language statements of an application program or by the RMS utility programs DEFINE and DESCRIBE. RMS uses this information about the attributes to structure a file on the storage medium.

The most important attribute of any RMS file is its organization. A file for use in a particular application can be tailored by making the proper selection of this and other attributes. In addition to file organization, the user can choose from among the following attributes:

- Storage medium on which the file resides.
- File and protection specification of the file.
- Format and size of records.
- Size of the file.
- Size of a particular storage structure, known as the bucket, within relative and indexed files.
- Definition of keys for indexed files.

Storage Media

Selection of a storage medium on which RMS builds a file is related to the organization of the file. Permanent sequential files can be created on disk devices or ANSI magnetic tape volumes. Transient files can be written on devices such as lineprinters and terminals. Unlike sequential files, relative and indexed files can reside only on disk devices.

File Specifications

The name assigned to a new file enables RMS to find the file on the storage medium. The file can be located on another node in a DECnet environment using a node specification. RMS allows for the assignment of a protection specification to a file at the time it is created.

When a file is created, the user must provide the format and maximum size specifications for the records the file will contain. The specified format establishes how each record appears physically in the file on a storage medium. The size specification allows RMS to verify that records written into the file do not exceed the length specified when the file was created.

RMS Record Formats

RMS record formats include fixed, variable, variable-with-fixed-control (VFC), and stream. Like the selection of a storage medium, the choice of a format for the records of a file depends on a file's organization. Table 18-2 shows the allowed combinations of record format and file organization.

Table 18-2 ▪ Record Formats and File Organizations

File Organization	Record Format			
	Fixed	Variable	VFC	Stream
Sequential	Yes	Yes	Yes	disk only
Relative	Yes	Yes	Yes	No
Indexed	Yes	Yes	No	No

- FIXED-LENGTH RECORD FORMAT

  The term "fixed-length record format" refers to records of a file that must all be one specified size. Each record occupies an identical amount of space in the file.

- VARIABLE-LENGTH RECORD FORMAT

  In variable-length record format, records in a file can be either equal or unequal in length. To allow retrieval of variable-length records from a file, RMS prefixes a count field to each record it writes. The count field describes the length (in bytes) of the record. RMS removes this count field before it passes a record to the program.

- VARIABLE-WITH-FIXED-CONTROL RECORD FORMAT

  Variable-with-fixed-control (VFC) records consist of two distinct parts, the fixed-control area and the user data record. The size of the fixed-control area is identical for all records of the file. The contents of each fixed-control area are completely under the control of the program and can be used for any purpose. As an example, fixed-control areas can be used to store the identifier (for example, relative record number or RFA) of related records.

  The second part of a VFC record is similar to a variable length record. It is a user data record, variable in length and composed of individual data fields.

- STREAM FORMAT RECORDS

  Records in stream format can vary in size. However, no count field precedes each record. Instead, RMS considers the entire file a stream of contiguous ASCII characters. Each record in the file is delimited by one of the following:

  - Formfeed (FF).
  - Vertical tab (VT).
  - Linefeed (LF).
  - Carriage return immediately followed by linefeed (CR-LF).

Stream format records are supported for file interchange with non-RMS application programs. Because this format is not very efficient, it should be used only when such interchange is a concern.

Size of Records

The programmer provides RMS with record size information along with the selected record format. RMS use of this information depends on the record format chosen.

When fixed-format records are chosen, the actual size of each record in the file must be indicated. This size specification becomes part of the information stored and maintained by RMS for the file. Thereafter, if a program attempts to write a record whose length differs from this specified size, RMS will reject the operation.

When creating a file with variable-length format records, you can specify a maximum record size greater than zero or, for sequential and indexed files, a maximum record size equal to zero. If the specified size is greater than zero, RMS interprets the value as the size of the largest record that can be written into the file.

VFC format records require two size specifications. The first size identifies the length of the fixed-control area of all records in the file; the second size specification represents the maximum length of the data portion of the VFC records. RMS handles this second size specification in a manner similar to its handling of the size specification for variable format records.

For stream format records, RMS permits the user to specify the same record size information as for variable format records. That is, a nonzero value represents the maximum permitted size of any record written in the file while a zero value suppresses RMS size checking.

Size of RMS Files

The size of an RMS file is expressed as a number of virtual blocks. Virtual blocks are physical storage structures. That is, each virtual block in a file is a unit of data whose size depends on the physical medium on which the file resides. The size of virtual blocks in files on disk devices, for example, is 512 bytes.

The operating system assigns ascending numbers to a file's virtual blocks. This numbering scheme allows a file to appear as a series of adjacent virtual blocks. In reality, though, the successive numbering of virtual blocks and the physical placement of these blocks on a storage medium need not correspond.

The virtual blocks of a file contain the records that programs write into the file. Depending on the size of records, a virtual block can contain one record, more than one record, or a portion of a record.

When creating an RMS file, you can specify an initial allocation size. If no file size information is given, RMS allocates the minimum amount of storage needed to contain the defined attributes of the file.

Buckets in Relative and Indexed Files

RMS uses a storage structure known as a bucket for building and maintaining relative and indexed files. Unlike a virtual block, a bucket can never contain a portion of a record. That is, RMS does not permit records to span bucket boundaries.

The size of buckets in a file is defined at the time the files are created. A large bucket size will serve to increase sequential mode processing speed of a file because fewer actual I/O transfers are required to access records. Minimizing bucket size, on the other hand, means that less I/O buffer space is required to support file processing.

Key Definitions for Indexed Fields

To define a key for an indexed file, the position and length of character data in the records of the file must be specified. At least one key, the primary key, must be defined for an indexed file. Additionally, up to 254 alternate keys can be defined. Each primary alternate key represents from 1 to 255 characters in each record of the file.

When identifying the position and the length of keys to RMS, you can define either simple or segmented keys. A simple key is a single, contiguous string of characters in the record; in other words, a single data field. A segmented key, however, can consist of from two to eight data fields within records. These data fields need not be contiguous, and RMS treats the separate data fields (segments) as a logically contiguous character string.

At file creation time, two characteristics for each key can be specified. Duplicate key values are allowed, or the key value can change. If duplicate key values are allowed, the programmer indicates that more than one record in the file can have the same value in a given key.

The personnel file can serve as an example of the use of duplicate keys. At file creation time, the department name field could be defined as an alternate key. As programs write records into the file, the alternate index for the department name key field would contain multiple entries for each key value (for example, PAYROLL, SALES, ADMINISTRATION) because departments are composed of more than one employee. When such duplication occurs, RMS stores the records so that they can be retrieved in first-in/first-out (FIFO) order.

An application could be written to list the names of employees in any particular department. A single execution of the application could, for example, list the names of all employees working in the department called SALES. By randomly accessing the file by alternate key and the key value SALES, the application would obtain the first record written into the file containing this value. Then, the application could switch to sequential access and successively obtain records with the same value, SALES, in the alternate key field. Part of the logic of the application would be to determine the point at which a sequentially accessed record no longer contained the value SALES in the alternate key field. The program could then switch back to random access mode and access the first record containing a different value (for example, PAYROLL) in the department name key field.

The second key characteristic (key value can change) indicates that records can be read and then written back into the file with a modified value in the key. When such modification occurs, the appropriate index is automatically updated to reflect the new key value. This characteristic can be specified only for alternate keys. Further, when specifying this characteristic, the user must also specify that the duplicate key values are allowed.

If the sample personnel file were created with the department name field as an alternate key, the creator of the file would need to specify that key values can change. This allows a program to access a record in the file and change the contents of a department name data field to reflect the transfer of an employee from one department to another.

The user can also declare the converse of either of these two key characteristics. That is, the user can specify for a given key that duplicate key values are not allowed or that key values cannot change. When duplicate key values are not allowed, RMS rejects any program request to write a record containing a value in the key that is already present in another record. Similarly, when the key value cannot change, RMS does not allow a program to write a record back into the file with a modified value in the key.

▪ RMS Processes Both Files and Records

After RMS has created a file according to the user's description of file characteristics, a program can access the file and store and retrieve data. The organization of the file determines the types of record operations permitted.

If the record accessing capabilities of RMS are not used, programs can access the file as a physical structure, in which case RMS considers the file simply as an array of virtual blocks. To process a file at the physical level, programs use a type of access known as *block I/O*.

File Processing

At the file level (that is, independent of record processing) a program can create, open, modify, extend, close, and delete a file. Once a program has opened a file for the first time, it has access to the unique internal ID for the file. If the program intends to open the file subsequently, it can use the internal ID to open the file. This avoids spending time on a directory search.

Directory Operations

RMS directory operations affect only directory entries, not the actual contents of the files. Directory operations include

- ENTER—create a directory entry.
- REMOVE—delete a directory entry.
- RENAME—replace a directory entry.
- PARSE—analyze a file specification.
- SEARCH—search directories.

On RSTS/E, the directory operations are RENAME, PARSE, and SEARCH.

File Operations

File operations affect files as whole entities rather than individual records or blocks in files. The file operations are

- CREATE—create a file (with a corresponding dictionary entry) and open the file for processing.
- OPEN—open an existing file for processing.
- DISPLAY—write file information to control blocks.
- ERASE—delete file contents (records or blocks) and remove dictionary entry.
- EXTEND—increase the allocation for a file.
- CLOSE—close an open file.

Record Operations on RMS Files

The organization of a file, defined when the file is created, determines the types of operations that the program can perform on records. Depending on file organization, RMS permits a program to perform the following record operations:

- Read a record—RMS returns an existing record within the file to the program.
- Write a record—RMS adds a new record that the program constructs to the file. The new record cannot replace an already existing record.
- Find a record—RMS locates an existing record in the file. It does not return the record to the program, but establishes a new current position in the file.
- Update a record—The program modifies the contents of a record read from the file. RMS writes the modified record into the file, replacing the old record.

Sequential File Organization Record Operations

In sequential file organization, a program can read existing records from the file using sequential or RFA access modes. New records can be added only to the end of the file and only through the use of sequential access mode. The find operation is supported in both sequential and RFA access mode. In sequential access mode, the program can use a find operation to skip records. In RFA access mode, the program can use the find operation to establish a random starting point in the file for sequential read operations. The sequential file organization does not support the delete operation because the structure of the file requires that records be adjacent in and across virtual blocks. A program can, however, update existing records in disk files as long as the modification of a record does not alter its size.

Relative File Organization Record Operations

Relative file organization permits programs greater flexibility in performing record operations than does sequential organization. A program can read existing records from the file using sequential, random, or RFA access mode. New records can be sequentially or randomly written as long as the intended record cell does not already contain a record. Similarly, any access mode can be used to perform a find operation. After a record has been found or read, RMS permits the delete operation. Once a record has been deleted, the record cell is available for a new record. A program can also update records in the file. If the format of the records is variable, update operations can modify record length up to the maximum size specified when the file was created.

Indexed File Organization Record Operations

Indexed file organization provides the greatest flexibility in performing record operations. A program can read existing records from the file in sequential, RFA, or random access mode. When reading records in random access mode, the program can choose one of four types of matches that RMS must perform using the program-provided key value. The four types of matches are

- Exact key match.
- Approximate key match.
- Generic key match.
- Approximate and generic key match.

Exact key match requires that the contents of the key in the record retrieved precisely match the key value specified in the program read operation.

The approximate match facility allows the program to select either of two relationships between the key of the record retrieved and the key value specified by the program. These are equal to or greater than. The advantage of this kind of match is that if the requested key value does not exist in any record of the file, RMS returns the record that contains the next higher key value. This allows the program to retrieve records without knowing an exact key value.

Generic key match means that the program need specify only an initial portion of the key value, thereby forming a logical truncation upon the key. RMS returns to the program the first occurrence of a record whose key contains a value beginning with those characters. This capability is useful in applications in which a series of records must be retrieved according to the contents of only a part of the key field. In an indexed inventory file, for example, a company might designate its part numbers in such a way that the first three digits represent the vendor from whom the part is purchased. In order to retrieve the record associated with a particular part, the program would normally supply the entire part number. Generic selection permits the retrieval of the first record representing parts purchased from a specific vendor.

The final type of key match combines both generic and approximate facilities. The program specifies only an initial portion of the key value, as with generic match. Additionally, a program specifies that the key data field of the record retrieved must be either equal to or greater than the program-supplied value or greater than the program-supplied value.

In addition to versatile read operations, RMS allows any number of new records to be written into an indexed file. It rejects a write operation only if the value contained in a key of the record violated a user-defined key characteristic (for example, duplicate key values are not permitted).

The find operation, similar to the read operation, can be performed in sequential, RFA, or random access mode. When finding records in random access mode, the program can specify any one of the four types of key matches provided for read operations.

In addition to read, write, and find operations, the program can delete any record in an indexed file and update any record. The only restriction RMS applies during an update operation is that the contents of the modified record must not violate any user-defined key characteristic (e.g., key values cannot change, and duplicate key values are not permitted).

Block I/O

Block I/O allows a program to bypass the record processing capabilities of RMS entirely. Rather than performing record operations through the use of supported access modes, a program can process a file as a physical structure consisting solely of virtual blocks.

Using block I/O, a program reads or writes multiple virtual blocks by identifying a starting virtual block number in the file. Regardless of the organization of the file, RMS accesses the identified block or blocks on behalf of the program.

Because RMS files, particularly relative and indexed files, contain internal information meaningful only to RMS itself, Digital does not recommend that a file be modified by using block I/O. The presence of the block I/O facility, however, does permit user-created file structures. The resultant structures must be user-maintained using specialized programs. The structures cannot be accessed using RMS record access mode and record operations.

▪ RMS Runtime Environment Has Two Levels

The environment within which a program processes RMS files at runtime consists of two levels: the file processing level and the record processing level.

At the file processing level, RMS and the host operating system provide an environment that permits concurrently executing programs to share access to the same file. RMS ascertains the amount of sharing permissible from information provided by the programs themselves. RMS also provides facilities that allow programs to minimize buffer space requirements for file processing.

At the record processing level, RMS allows programs to access records in a file through one or more record access streams. Each record access stream represents an independent and simultaneously active series of record operations directed toward the file. Within each stream, programs can perform record operations synchronously or asynchronously on operating systems that support this facility. That is, RMS allows programs to choose between receiving control only after a record operation request has been satisfied (synchronous operation) or receiving control before the request has been satisfied (asynchronous operation).

For both synchronous and asynchronous record operations, RMS provides two record transfer modes, move mode and locate mode. Move mode causes RMS to copy a record from an I/O buffer into a program-provided location. Locate mode allows programs to address records directly in an I/O buffer.

If your system has suitable DECnet facilities, RMS offers file and record access to files residing on other network nodes, providing the other nodes have an RMS-based File Access Listener (FAL). Generally, remote access is indistinguishable from local access. However, certain functions cannot be executed remotely. These include wildcard support, $PARSE, $SEARCH, $ENTER, $REMOVE, $RENAME, and transmission of device, directory, and file identifiers.

▪ The Processing Environment Allows Executing Programs to Share Files

RMS provides two major facilities at the file processing level, file sharing and buffer handling.

File Sharing

Timely access to critical files requires that more than one concurrently executing program be allowed to process the same file at the same time. Therefore, RMS allows executing programs to share files rather than requiring them to process files serially. The manner in which a file can be shared depends on the organization of the file. Program-provided information further establishes the degree of sharing of a particular file.

- FILE ORGANIZATION AND SHARING
  With the exception of magnetic tape files, which cannot be shared, every RMS file can be shared by any number of programs that are reading, but not writing, the file. Sequential files on disk can be accessed by a single writer or shared by multiple readers. Relative and indexed files, however, can be shared by multiple readers and multiple writers.

- PROGRAM SHARING
  A file's organization establishes whether it can be shared for reading with a single writer or for multiple readers and writers. A program specifies whether such sharing actually occurs at runtime. The user controls the sharing of a file through information the program provides RMS when it opens the file. First, a program must declare what operations it intends to perform on the file. Second, a program must specify whether other programs can read the file or both read and write the file concurrently with that program.

The combination of these two types of information allows RMS to determine if multiple user programs can access a file at the same time. Whenever a program's sharing information is compatible with the corresponding information another program provides, concurrent access is allowed.

- BUCKET LOCKING

RMS uses a bucket-locking facility to control operations to a relative or indexed file that is being accessed by one or more writers. The purpose of this facility is to ensure that a program can add, delete, or modify a record in a file without another program's simultaneously accessing the same record.

When a program opens an indexed or relative file with the declared intention of writing or updating records, RMS locks any bucket accessed by the program. This locking prevents another program from accessing any record in the bucket until the program releases it, and remains in effect until the program accesses another bucket. RMS then unlocks the first bucket and locks the second.

Buffer Handling

To a program, record processing under RMS appears as the movement of records directly between a file and the program itself. Transparently to the program, however, RMS reads or writes virtual blocks or buckets of a file into or from internal memory areas known as I/O buffers. Records within these buffers are then made available to the program.

In addition to buffers that contain virtual blocks or buckets, RMS requires a set of internal control structures to support file processing. The combination of these buffers and control structures is known as the space pool.

▪ Programmers Access Records through the RMS Record Processing Environment

After opening a file, a program can access records in the file through the RMS record processing environment. This environment provides three facilities:

- Record access streams.
- Synchronous or asynchronous record operations.
- Record transfer modes.

Record Access Streams

In the record processing environment, a program accesses records in a file through a record access stream, a serial sequence of record operation requests. For example, a program can issue a read request for a particular record, receive the record from RMS, modify the contents of the record, and then issue an update request that causes RMS to write the record back into the file. The sequence of read and update record operation requests can then be performed for a different record, or other record operations can be performed, again in a serial fashion. Thus, within a record access stream, there is at most one record being processed at any time. However, for relative and indexed files, RMS permits a program to establish multiple record access streams for record operations to the same file. The presence of such multiple record access streams allows programs to process in parallel more than one record of a file. Each stream represents an independent and concurrently active sequence of record operations. Further, when such streams update records in the file, RMS employs the same bucket-locking mechanism among streams that it uses to control the sharing of a file among separate programs.

Synchronous and Asynchronous Record Operations

Within each record access stream, a program can perform any record operation either synchronously or asynchronously. (RSTS/E RMS-11 supports synchronous record operations only.) When a record operation is performed synchronously, RMS returns control to a program only after the record operation request has been satisfied (for example, a record has been read and passed to one program). When a record operation is performed asynchronously, RMS can return control to one program before the record operations request has been satisfied. A program, then, can utilize the time required for the physical transfer between the file and memory of the block or bucket containing the record to perform other computations. However, a program cannot issue a second record operation through the same stream until the first record operation has completed. To ascertain when a record operation has actually been performed, a program can issue a wait request and regain control when the record operation is complete.

Record Transfer Modes

In addition to specifying synchronous or asynchronous operations for each request in a record access stream, a program can utilize either of two record transfers modes to gain access to each record in memory:

- Move Mode Record Transfers—RMS permits move-mode record operations for all file organizations and record operations. Move mode requires that an individual record be copied between the I/O buffer and a program. For read operations, RMS reads a block or bucket into an I/O buffer, finds the desired record within the buffer, and moves the record to a program-specified location.

  Before a write or update operation in move mode, the program builds or modifies a record in its own work space. Then the program issues a write or update record operation request, and RMS moves the record to an I/O buffer.

- Locate Mode Record Transfers—RMS supports locate-mode record transfers for read operations to all file organizations. However, it permits locate mode on write operations for sequential files only.

  Locate mode reduces the amount of data movement, thereby saving processing time. This mode enables programs to access records directly in an I/O buffer. Therefore, there is normally no need for RMS to copy records from the I/O buffer to a program. To allow the program to access a record in the I/O buffer, RMS provides the program with the address and size of the record in the I/O buffer.

Utilities Aid in RMS Development

There are a host of RMS utilities to help system managers and operators develop record management services. The following summarizes those utilities available.

File Design Utility (RMSDES)

The File Design Utility lets you interactively design and create RMS-11 files.

Index File Load Utility (RMSIFL)

This utility reads records from any type of RMS-11 file and loads them into an indexed file created especially to store those records. RMSIFL also compresses indexed files and converts ASCII stream files to RMS-11 indexed files.

File Conversion Utility (RMSCNV)

File Conversion reads records from an RMS-11 file of any type and loads them into another RMS-11 file of any type. File conversions can also take place over networks on operating systems with DECnet network capabilities and Data Access Protocol (RMSDAP) support.

File Backup Utility (RMSBCK)

This utility copies the contents of any RMS-11 disk file to another disk or to a magtape container file created for backup purposes.

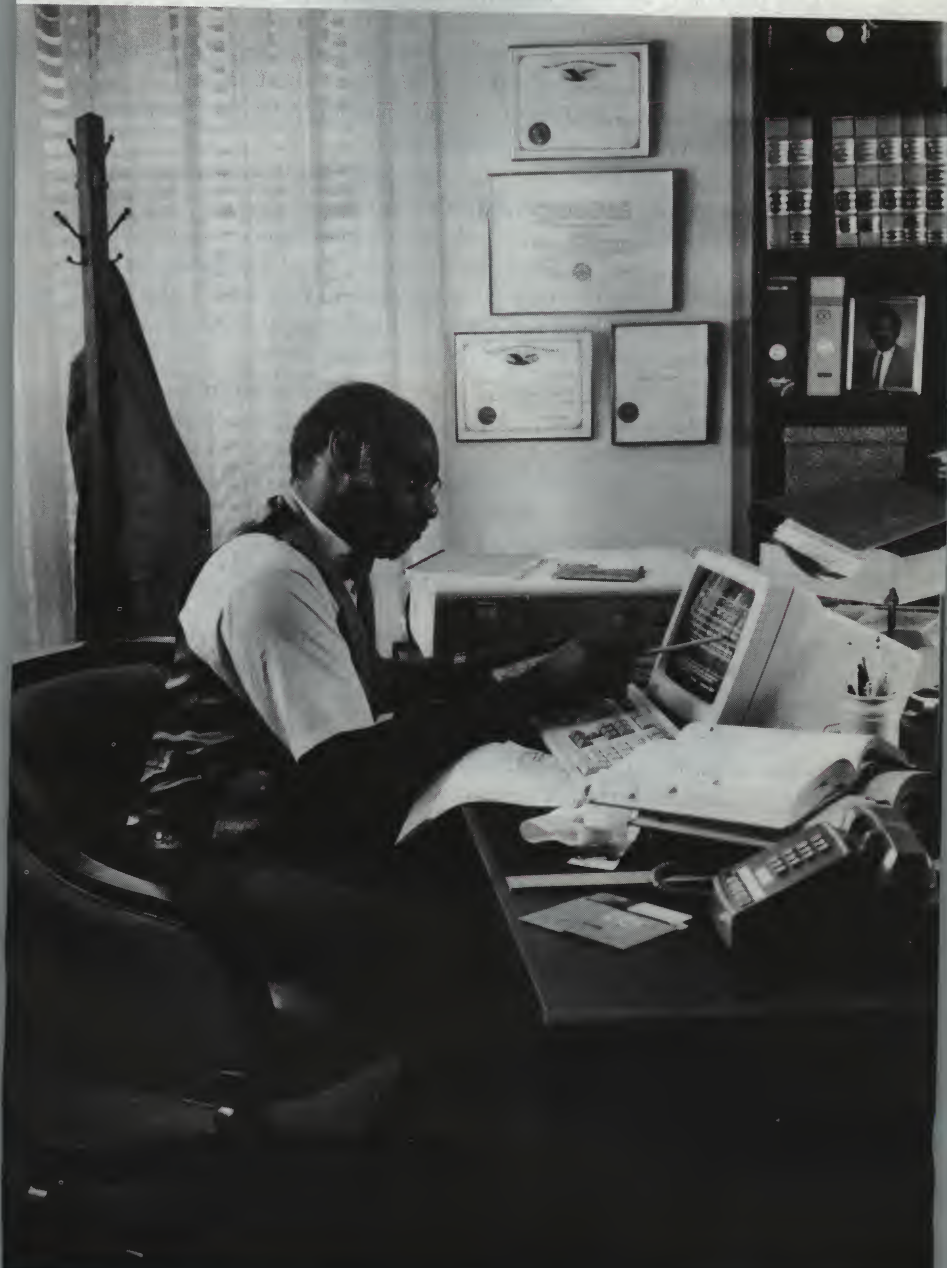
File Display Utility (RMSDSP)

This utility provides a concise description of any RMS-11 file, including backup container files.

File Restoration Utility (RMSRST)

The File Restoration Utility restores to disk those files that were backed up on magnetic tape or disk by the Backup Utility. This utility allows for selective restoration of any account to which you have access privileges and includes features providing data integrity checks that let you check the reliability of the restored data files. In addition, it provides for backup and restore intersystem transportability.

Chapter 19 ▪ DATATRIEVE-11



DATATRIEVE-11 Puts Information at Your Fingertips

DATATRIEVE-11 is a data maintenance, inquiry, and report writing system available in the PDP-11 family on RSX-11M-PLUS, RSX-11M, Micro/RSX, RSTS/E, Micro/RSTS, IAS, and P/OS operating systems. It provides users with direct, fast, easy access to the data in sequential, indexed, and relative RMS-11 files. DATATRIEVE-11 accepts simple words and phrases to extract, modify, or update RMS-11 data. With fewer than ten commands, users can find, print, update, and sort records.

By eliminating the need for many specialized application programs and their time-consuming compilations, DATATRIEVE-11 helps to maximize programmer and system productivity. And thanks to its familiar syntax, DATATRIEVE-11 is easy to learn and simple to use. You can access data without the services of a programmer. In addition, it immediately notifies you of any errors so that you can correct them immediately. DATATRIEVE-11 makes file access simple, while maintaining the file security provided by the operating system.

Designed to be used by both novices and computer professionals, DATATRIEVE-11 operates effectively in commercial, technical, scientific, industrial, or educational environments. Typical applications range all the way from producing a complex report to answering a casual question. For example, using DATATRIEVE-11, a personnel file could be queried to determine how many employees held bachelor's degrees, or the same file could be used to produce a report with a complete statistical analysis of the employee education versus compensation.

Another typical environment where DATATRIEVE-11 would be useful is a distributorship with an order processing system. In this setting, sales data could be extracted by territory. Order backlogs might be retrieved, sorted by supplier, and printed on a report.

Additional facilities are provided by the system for selective data retrieval, sorting, formatting, updating, and report generation.

▪ DATATRIEVE-11 Helps You Produce Detailed Reports

There are many advantages to using DATATRIEVE-11 over application programs when generating ad hoc queries and reports. The three major categories of DATATRIEVE-11 capabilities are

- Data access and update facilities
- Report-generation facilities
- Data dictionary facility

Data Access and Update Facilities

The INQUIRY (data access) and UPDATE commands provided by DATATRIEVE-11 let you manipulate records and files. DATATRIEVE-11 offers simple and advanced commands. Simple commands enable the novice to find, update, and sort records. Advanced commands can be used to perform more complex functions such as combining commands to form procedures.

DATATRIEVE-11 provides the first-time user with flexible "value-based" data access/update capabilities that can eliminate the need for programming overhead in many situations. Information is returned to the user in the form of collections of records that can be manipulated and/or displayed on the terminal or printer using the DATATRIEVE-11 report writing facility. Several specific features bring this power to users.

- GUIDE MODE is a tutorial aid with automatic prompting. This feature permits the novice to retrieve and display data by stepping through a subset of commands.
- The documentation set for DATATRIEVE-11 includes a Primer, designed to introduce the novice to the use of DATATRIEVE-11, and a User's Guide that uses examples to present the various DATATRIEVE-11 functions.
- The commands are simple words and phrases instead of confusing acronyms.
- DATATRIEVE-11 supports simple arrays.
- A data type is provided that recognizes data formats and facilities, entering and displaying data in any one of several formats.
- DATATRIEVE-11 provides a full set of arithmetic operators (addition, subtraction, multiplication, division, and negation), statistical operators (total, average, maximum, minimum, and count), and conversion between data types used in Digital's FORTRAN, COBOL, DIBOL, and BASIC-PLUS-2 languages.

Report Generation

In addition to its inquiry and update commands, DATATRIEVE-11 provides a report writing facility to generate reports from RMS-11 files. The data can come directly from the files or can be preselected and manipulated through a series of DATATRIEVE-11 commands. Users can specify such parameters as spacing, titles, headings, and totals on their reports. As in the inquiry and update facility, errors in commands are discovered immediately to avoid printing wrong or incomplete reports.

When reporting requirements change, you need not rewrite or modify an entire reporting program. All you do is issue modified statements to the report facility. The DATATRIEVE-11 report writer provides easy-to-use commands to control the following report functions:

- Report name, date, and page numbering.
- Page width and length specification.
- Detail line specification.
- Multiple control break specification with automatic totaling at any level.
- Multiple report sections.
- Statistical lines—such as total, average, and count.

A DATATRIEVE-11 report command can be freely intermixed with other DATATRIEVE commands.

Data Dictionary Facility

The Data Dictionary maintains definitions of record structures and domain names. A record structure describes the format of the records in the file. A domain is a named group of data containing records of a single type. Record structures and domain names must be defined before DATATRIEVE-11 can be used to access data.

The definitions provide a substantial level of data and program independence because the definitions (or views) can cross file boundaries. Thus, by providing a single value-based DATATRIEVE-11 query, users can access information from multiple files and records. DATATRIEVE-11 also provides commands to list the contents of the Data Dictionary, to delete entries, and to control access to individual entries.

▪ DATATRIEVE-11 Commands Let You Store, Update, and Retrieve Information

DATATRIEVE-11 is a multifaceted data management facility that uses a set of English commands for data retrieval, modification, display, and report generation. Prompting is automatic for both command and data entry. The major commands include

- HELP—provides a summary of each DATATRIEVE-11 command.
- READY—identifies a domain for processing and controls the access mode to the appropriate file.
- FIND—establishes a collection (subset) of records contained in either a domain or a previously established collection based on a Boolean expression.
- SORT—re-orders a collection of records in either the ascending or descending sequence of the contents of one or more fields in the records.
- PRINT—prints one or more fields of one or more records. Output can optionally be directed to a lineprinter or disk file. Format control can be specified. A column header is generated automatically.
- SELECT—identifies a single record in a collection for subsequent individual processing.
- MODIFY—alters the values of one or more fields for either the select record or all records in collection. Replacement values are prompted for by name.
- STORE—creates a new record. The value for each field contained in the record is prompted for by name or indicated on a predefined form.
- ERASE—deletes one or more records from the RMS-11 file corresponding to the appropriate domain.
- FOR—executes a subsequent command once for each record in record collection, providing a simple looping facility.
- DECLARE—defines global and local variables to be used within a DATATRIEVE-11 query.
- DEFINE—provides a consistent mechanism for creating domain, record, table, procedure, and view definitions in the DATATRIEVE-11 Data Dictionary.
- EDIT—invokes an editor that inserts, modifies, or deletes text from the DATATRIEVE-11 Data Dictionary

In addition to the simple data manipulation commands, a number of more complex commands are available for the advanced user. These commands, such as REPEAT, BEGIN-END, and IF-THEN-ELSE, may be used to combine two or more DATATRIEVE-11 commands into a single compound command. These, in turn, may be stored in the Data Dictionary as procedures for invocation by less experienced users. DATATRIEVE-11 can be used interactively from a terminal or in batch mode.

Data Definition

The data definition process involves establishing special DATATRIEVE-11 constructs called *domains*. The domain concept is central to DATATRIEVE-11. Domains represent relationships between actual physical data and descriptions of data. DATATRIEVE-11 performs all data management in terms of domains. Domains must be defined before DATATRIEVE-11 can manage the data associated with them.

In the simplest form, a DATATRIEVE-11 domain definition consists of a domain name, the name of the RMS-11 file, and the name of a record format description. A record format description defines the fields within a record, assigning each field a name and specifying its length, data type, and other vital parameters. All DATATRIEVE-11 domain definitions and record format descriptions are contained in the DATATRIEVE-11 Data Dictionary.

Record format descriptions can specify data validation criteria on a per-field basis. DATATRIEVE-11 automatically uses the validation parameters to screen data at the time of input so that only data defined as valid is accepted. Supported validation parameters include range checks and must-match tables.

Domains can span multiple RMS-11 files and can also include the name of an associated DATATRIEVE-11 table.

Data Management

Data management involves creating and maintaining data in a current and correct state by adding, eliminating, and modifying records. The STORE, ERASE, and MODIFY statements are used to perform these relatively straightforward functions.

When an application requires the creation of new files, the files must be filled with data. This process is called "populating" the file. A series of successive STORE statements is used for this purpose. With the STORE statement, DATATRIEVE-11 prompts the user for each field value and, before accepting input, performs any validation checks specified by the record format description.

Data Retrieval

Maintaining an accurate database, however, is not an end in itself. Data is used to make decisions, generate reports, initiate transactions, and generally facilitate the operational processes of an enterprise. DATATRIEVE-11 allows stored data to be retrieved in an easily understood form regardless of the underlying data structure.

The data retrieval statements of DATATRIEVE-11 are simple and particularly powerful statements. They consist of verbs modified by a Record Selection Expression (RSE). The RSE defines a subset of the records in the domain. These records are then selected by DATATRIEVE-11 for retrieval. One statement can get the answer to a casual query or produce a long detailed report.

"EMPLOYEES WITH SALARY GREATER THAN $20,000," "ACCOUNTS WITH UNPAID-BALANCE GREATER THAN $600," or "DONORS WITH BLOOD TYPE EQUAL O-NEG" are examples of typical RSEs. Multiple conditions can be combined in a single RSE – for example, "DONORS WITH BLOOD TYPE EQUAL O-NEG AND LAST DONATION DATE LESS THAN JANUARY 1982." The DATATRIEVE-11 SORT operator can be appended to the RSE to order the records being retrieved.

Ad hoc information retrieval with DATATRIEVE-11 is normally performed as an iterative process using a series of statements to progressively narrow down the group of records to be retrieved. This works by using a FIND request with a specified domain as its object to establish what is called the current collection. Subsequent FIND requests progressively narrow down the current collection until the user is satisfied with the results. For example, the statement "FIND DONORS WITH BLOOD TYPE EQUAL O-NEG AND LAST DONATION DATE LESS THAN JANUARY 1981" might yield the DATATRIEVE-11 response "200 RECORDS FOUND." In this case, the user could narrow down the current collection with the statement "FIND CURRENT WITH ZIP CODE EQUAL 77451." DATATRIEVE-11 might then respond with "14 RECORDS FOUND" and the user could print these records to get telephone numbers for soliciting blood donations to help an accident victim.

Reports

The PRINT statement is used to output information to a display terminal, a printer, or an RMS-11 file. Though there are some formatting options possible with the PRINT statement, they are limited. The REPORT command provides a more comprehensive set of formatting options for producing standard printed reports with page and column headings, page numbers, totals, and subtotals.

Stored Procedures

With the DEFINE PROCEDURE command, users can define sequences of DATATRIEVE-11 commands and statements and store them for later use. PROCEDURES can be invoked to be run by themselves or can be embedded in other sequences of commands and statements.

▪ DATATRIEVE-11 Is Easy to Use

DATATRIEVE-11 ease-of-use features include a guide mode, an editor, and an *Application Design Tool.*

Guide Mode

DATATRIEVE-11 provides a self-teaching facility, called *guide mode,* for use with VT52 and VT100 family terminals. In this mode of operation, users are guided through their DATATRIEVE-11 sessions with a series of prompts.

To invoke guide mode, the user issues a SET GUIDE command. DATATRIEVE-11 immediately responds with "ENTER COMMAND, TYPE ? FOR HELP." If "?" is typed at this point, DATATRIEVE-11 presents the user with the possible responses—in this case, READY, SHOW, or LEAVE. If one of the alternatives is selected, DATATRIEVE-11 then proceeds to guide the user through the syntax of the selected statement. In the case of READY, DATATRIEVE-11 prompts with "DOMAIN NAME, END WITH SPACE."

DATATRIEVE-11 Editor

The DATATRIEVE-11 editor is a subset of the Digital standard editor, EDT. It can be used only in line mode and can edit only definitions stored in the DATATRIEVE Data Dictionary.

Application Design Tool

The Application Design Tool (ADT) is a DATATRIEVE-11 utility that simplifies the process of defining domains, record formats, and creating RMS-11 data files. Operating in an interactive mode, ADT presents the user with a series of simple questions. The user's responses provide ADT with information to generate the proper definitions. For RMS-11 files, ADT will prompt the user to get a full set of parameters pertaining to organization, index keys, etc. ADT will then create an indirect command file that the user can execute immediately or at some later time to create the desired file.

▪ DATATRIEVE-11 Comes with Advanced Features, Too

DATATRIEVE-11 allows domains to be defined that can subset the fields of a record and can span multiple RMS-11 files. These are called *view domains* because they provide a user's logical view of the data. Once view domains have been established, they can be used in much the same way as simple domains.

This facility is basic to high-level data access. It makes it possible for a single statement to retrieve a set of related records. For example, in an employee records application there might be an employee master file with company confidential information pertaining to salary that could be masked out during a view domain. Other information in the master file such as addresses and telephone numbers could then be combined in another view domain with a special file of records used in a car-pooling application.

View domains can also be used with RMS-11 files for domains containing records related in a hierarchical fashion. For example, in an order processing application there might be an account master file and an order file. These files could be combined in a view to produce billing statements with data drawn from both files. A single record in this view domain could be defined to contain one account master record and all the orders applying to that account.

DATATRIEVE-11 *tables* can be defined to reside in the DATATRIEVE-11 Data Dictionary. Tables can be used as a must-match list for field validation or for argument function type conversions. For instance, a must-match list of valid U.S. Postal Service state abbreviations could be used to check an address field, or an argument function table could be used to convert from state abbreviation codes to the spelled out state name.

▪ DATATRIEVE-11 Has Its Own Protection System

Data protection is accomplished through two independent mechanisms — the protection systems of the host operating system and those within DATATRIEVE-11. The DATATRIEVE-11 protection system uses passwords and User Identification Codes (UICs or PPNs) to allow a user to regulate access to domains, records, procedures, and tables through access requirements recorded in the Data Dictionary. Thus, each resource has its own security system to assure access is not granted to unauthorized users.

DATATRIEVE-11 Version 3 Offers Users a Full Range of Data Access

The latest edition of DATATRIEVE-11 is version 3, which is well suited to low-cost, multiuser systems such as the MicroPDP-11. Version 3 gives users access to data not only on their own systems but on departmental or corporate PDP-11 and VAX systems as well. Users at all levels can also write entire applications exclusively in DATATRIEVE.

DATATRIEVE-11 Version 3 features
▪ A new distributed information management facility that allows access to DATATRIEVE domains on a PDP-11 system from a VAX system via DECnet. As a result, VAX users can transparently access data, whether on their own system, another VAX, or a PDP-11 running DATATRIEVE.
▪ A new Remote Call Interface that allows PDP-11 users to access domains on other PDP-11s or VAX systems. Users can write programs in COBOL, BASIC, or FORTRAN to access remote DATATRIEVE domains.
▪ A new DROP statement that allows users to drop selected records from a collection. Users can use DROP to refine a collection until it contains exactly the records they want.
▪ A startup command file capability that provides users with a means of executing DATATRIEVE commands and statements automatically, everytime they call DATATRIEVE.
▪ A capability for users to store comments as well as definitions in the Data Dictionary when defining dictionary objects.
▪ Improved disk space for Data Dictionaries as a result of reduction of record and bucket size.
▪ Conversion of hyphens to underscores for greater compatibility with VAX DATATRIEVE.

PRO/RDT

PRO/RDT is an information management tool that works with PRO/DATATRIEVE to let the user extract selected information from a DATATRIEVE domain on a remote host. This host can be another PDP-11 system, VAX, Professional, or DECsystem-20. The remote host must have DECnet or PRO/DECnet and a DATATRIEVE Distributed Data Management Facility (DDMF) task installed. PRO/DDMF, which is packaged with the PRO/RDT application, lets the Professional be a remote host for other PDP-11, PRO, VAX, or DECsystem-20 systems. PRO/RDT uses DECnet as the transfer medium. DATATRIEVE and its protocols control the information transfer.

PRO/RDT is menu-driven and contains extensive online help. PRO/RDT menus prompt the user for all information needed to create and process an abstract description or extract. An extract identifies the DATATRIEVE data to be retrieved. This information includes system node name, user name, password, dictionary, and domain name. The user is also prompted for the name of the dictionary domain and data file where the extracted information will be stored.

Through the PRO/RDT Main Menu the user can

- Create a new extract.
- Get an old extract.
- Process the current extract.
- Change the current extract.
- Save the current extract.
- Run PRO/DATATRIEVE.
- View the P/OS message/status board.

Through a series of submenus, users can establish various selection criteria (for example, records and fields to be extracted, sort sequences, and key selection). In this way, some or all of the information in a remote domain can be brought over to the user's system.

When PRO/RDT processes an extract, information is extracted from a remote host and stored in the user's file. Domain and record definitions for the extracted information are stored in a PRO/DATATRIEVE dictionary. The user can then select PRO/DATATRIEVE from the PRO/RDT Main Menu to manipulate the extracted information and create reports. Stored extracts can be processed repeatedly to retrieve updated information from a remote host.

PRO/RDT can extract information from any remote DATATRIEVE domain. However, PDO/RDT will not process data within lists. Thus, if the remote domain is not "flat," the user may access data in the fixed part of the record only. PRO/RDT cannot retrieve COMPUTED BY or REDEFINES fields.

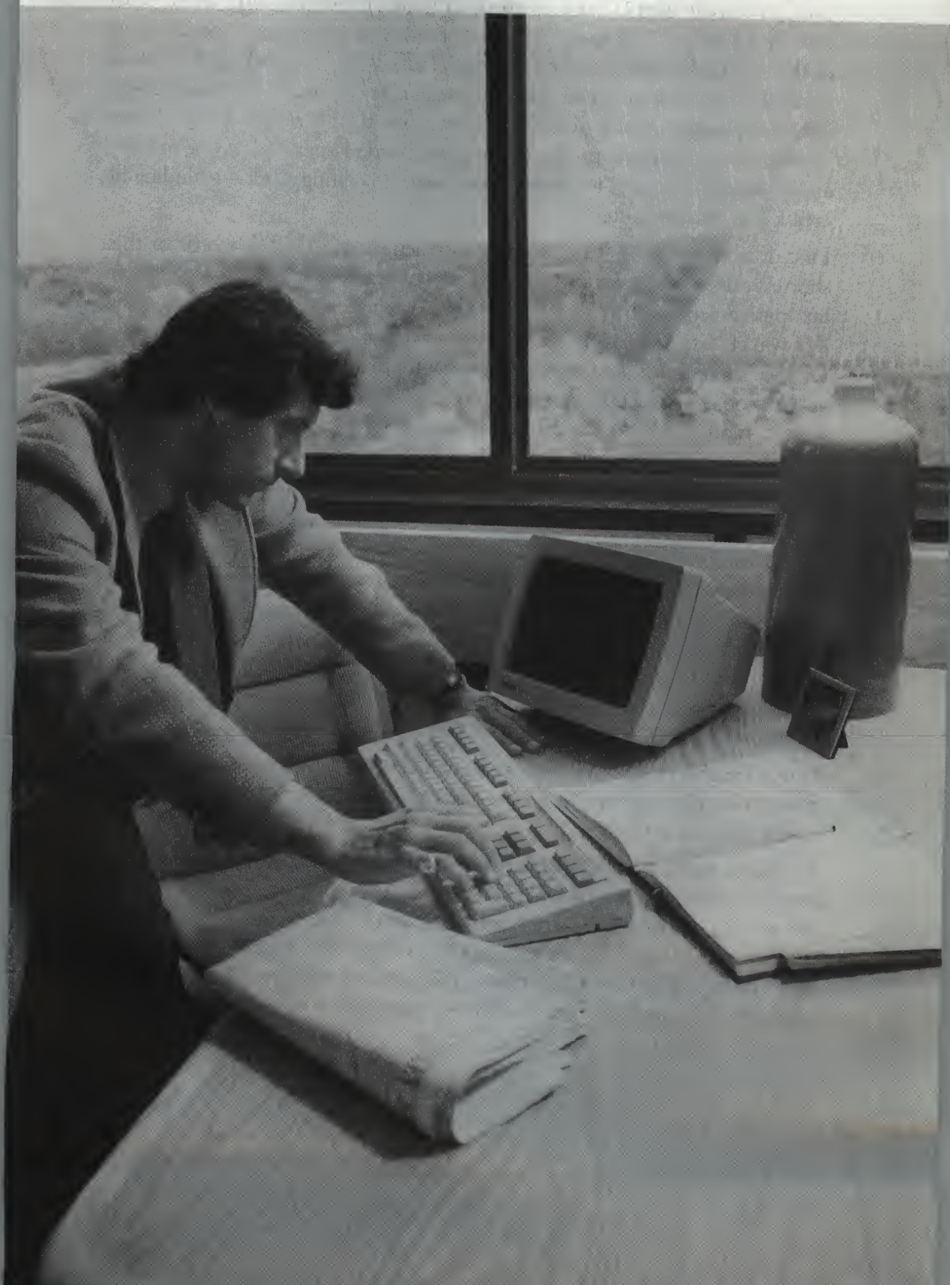
[illegible]

[illegible]

[illegible]

[illegible]

Chapter 20 ▪ The EDT Editor



▪ EDT Is Powerful, Yet Easy to Learn

EDT is a powerful text editor available on many Digital operating systems. Though you can use EDT at a variety of terminals, it is especially powerful when used on VT200, VT100, and VT52 videoterminals because it can take advantage of the editing keypad that these terminals offer. With keypad mode, a single keystroke performs an entire editing function, for example, deleting, reinserting, or replacing a word. You can even redefine the functions of keypad keys (through key macros) to produce commonly needed operations. EDT provides a wide range of benefits to anyone who has to do editing work—including file creation—on a computer.

First, it is very easy to learn. Editing instructions are English words or their shortest unique abbreviations. The order of operands is logical for English syntax; parameters can be either line numbers in the text file or character strings that you choose. Extensive facilities remind you quickly of the possible options for a particular command and of the format for that editing instruction. You can get help on general EDT operations by typing HELP. If you need help while in keypad mode, pressing the HELP key displays information that is specific to keypad editing.

Second, the editor protects your editing session with a journaling capability. EDT makes a record of everything you type so that your work will not be lost if your editing session is terminated by equipment failure. In addition, the editing session does not alter the original file until you are sure that you have done what you want. Instead, all editing activity takes place upon a copy of the original file in a temporary workspace called a buffer. A buffer is a part of EDT's memory that can hold an essentially unlimited amount of text. Only when you have ended the session do you have to determine whether or not to incorporate your editing activities into the file.

EDT is capable of working with many files at once. If you want to concatenate several files, create several files from one, or distribute part of a file among many others, you can do so with EDT.

In change mode on a VT200, VT100, or VT52 terminal, you edit one 22-line window (screenful) at a time so that you can observe immediately the effects of any editing operations you perform. Instead of being restricted to the most recently altered line, you can see a whole screenful of text, and see the relationship of new and old lines. Of course, if the text is longer than 22 lines, you can easily scroll through it to get to any point you want to edit.

▪ EDT Gives You a Choice of Editing Methods

With EDT you have a choice of keypad or line mode editing. These modes of editing allow you to

- Display a range of lines.
- Find, substitute, insert, and delete text.
- Move, copy, and renumber lines.
- Copy text into a buffer and write it on files.
- Define the functions of keys.

With keypad editing, you use the group of keys at the right of the keyboard to enter keypad functions. Keypad editing is powerful and versatile, yet it is easy to learn and to use. In keypad editing, the active buffer is displayed on the screen as you edit. You can see the changes you make to a buffer as they take place. There is a wide variety of keypad editing functions, each of which requires you to press only one or two keypad keys to perform a function. You enter commands, insert text, and perform control functions from the keyboard.

Line editing is useful for those who have hardcopy terminals or who prefer editing by numbered lines. In line editing, you make all entries from the keyboard. As you make changes to the contents of the buffer, EDT displays one or more lines at a time.

Keypad Layout

Keypad functions let you perform a variety of operations with a single keystroke. You can change the function of any keypad key to meet your needs with the DEFINE KEY command. Figure 20-1 shows the keypad for a VT100.

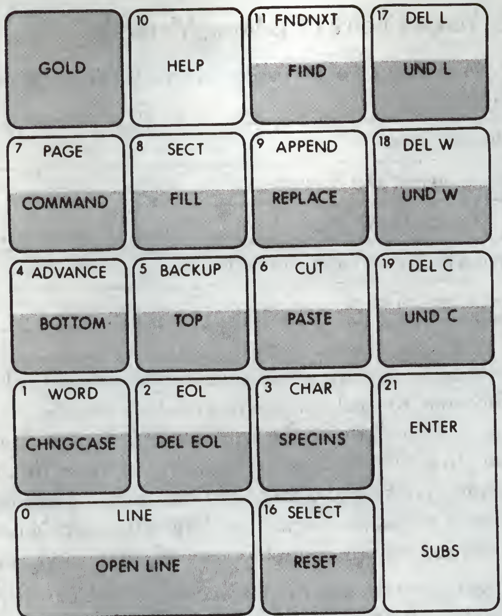


Figure 20-1 ▪ EDT VT100 Keypad Common Keyboard Functions

NOTE

The numbers in the upper-left corner of the keys are what actually appear on the keys.

Key	**Function**
Backspace	Go to beginning of line
Delete	Delete character
Linefeed	Delete to start of word
CTRL/A	Compute tab level
CTRL/D	Decrease tab level
CTRL/E	Increase tab level
CTRL/K	Define key
CTRL/T	Adjust tabs
CTRL/U	Delete to start of line
CTRL/W	Refresh screen
CTRL/Z	Return to line mode

Editing Operations

EDT lets you position the cursor, insert and delete text, reposition blocks of text within a file, create auxiliary files, and more. The combination of a clear and readable videodisplay screen and the extensive keypad functions supported by the videoterminals makes editing quick and easy.

- CURSOR POSITIONING

  You move the cursor around on the screen to position it properly for inserting or modifying text. The cursor can go in any direction. The keys, for example, let you move the cursor to the right or left by any number of characters or up and down by any number of lines. In addition, special function keys on the keypad move the cursor to the right or left by one word or one character, to the beginning or end of a line, or to the beginning or end of the buffer.

  You can also move the cursor through a buffer by specifying a character string that will serve as the object of a search. EDT then moves the cursor backward or forward directly to that point. The entire buffer is always available for editing; you may scroll forward or backward through the buffer at will.

- INSERTING AND DELETING TEXT

  There are several ways to insert and delete text in a buffer. You enter text by typing alphanumeric and special characters at the keyboard. You can delete a single character, a word, or a line, and multiple words and lines backward and forward relative to the cursor by using keypad functions. Furthermore, text deleted during the current editing session can be restored by using the UNDELETE keys to recall it from special buffers reserved for that purpose. This allows quick recovery from editing mistakes or mistyped commands. You can also combine insert and delete operations by using the special keypad functions for finding and substituting text.

- MOVING TEXT

  Special function keys on the keypad allow you to mark off an entire section of text and then move it to a new position in the file or string it together with other sections similarly marked off.

- CREATING AUXILIARY FILES

  From EDT's change mode you have access to EDT's line mode commands, including the WRITE and INCLUDE commands, which allow you to read and write files during an editing session.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

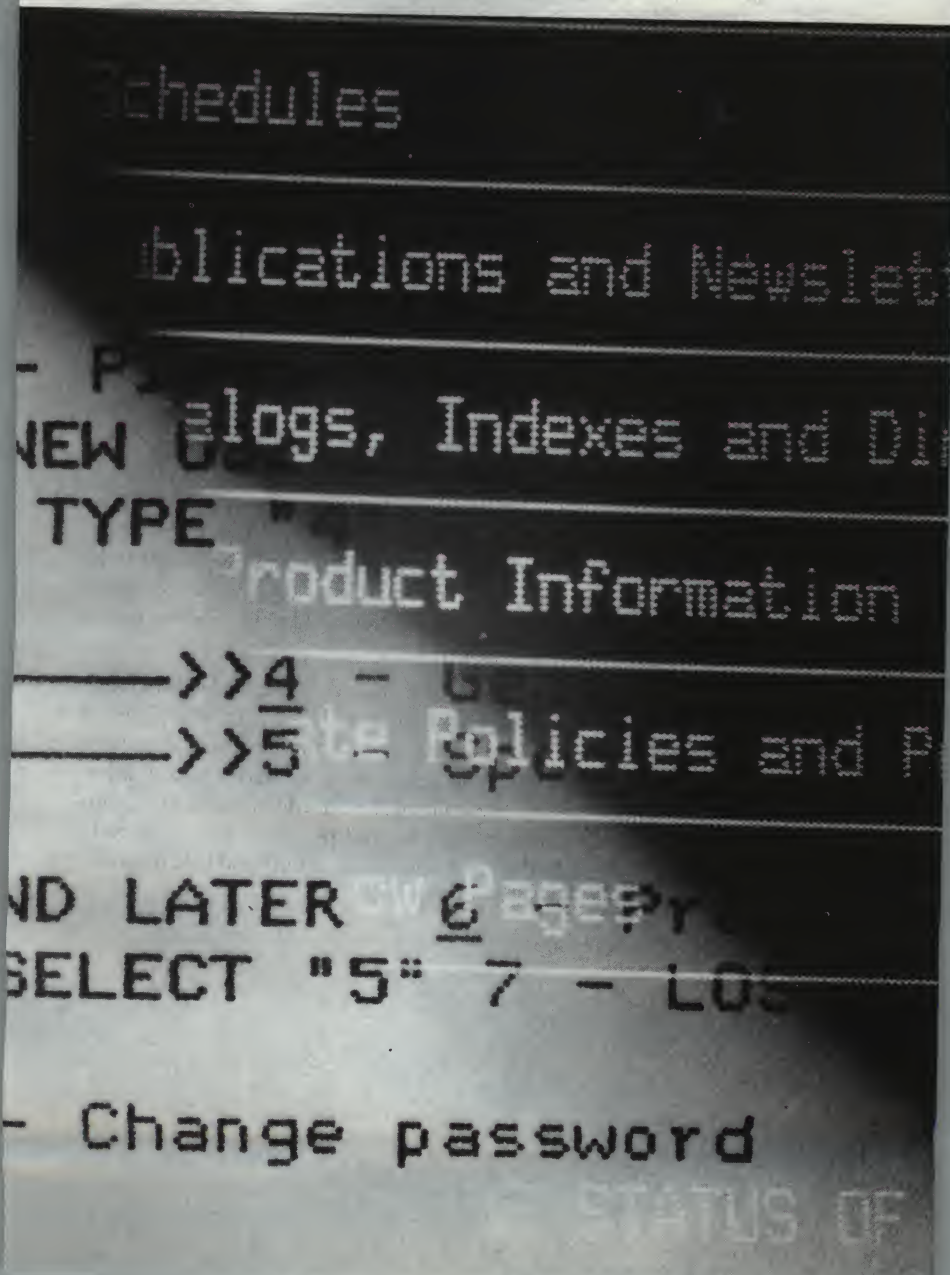
[illegible]

[illegible]

[illegible]

[illegible]

Chapter 21 • Screen Formatters



▪ FMS-11 Provides Sophisticated Screen Formatting to Applications

There are numerous instances of commercial, scientific, and industrial applications in which a formatted videoscreen makes the application so much easier to use. Clerks updating inventory lists, for example, can use a well-designed form to guarantee that the proper part codes, quantities, prices, suppliers, and other pertinent data are all entered exactly as required by the program that is manipulating them. Rather than presenting the clerk with a blank screen and a complicated menu of instructions for entering the data, a form-managed videoterminal supplies pictures that automatically indicate where each field goes, instantly checks to see that it is filled with the right number and types of characters, and aids the clerk with concise messages appropriate to the field or to the form as a whole.

Such forms can be extremely simple, or they can be quite complex, depending on the necessities of the program that uses the information being formatted. A chain of related forms might be required in some applications.

FMS-11 Forms Management System provides sophisticated screen formatting for application programs. By using FMS-11, it is easy to create and update video forms with the VT52 and VT-100 families of terminals. (FMS is not supported on the VT200 terminal.) FMS-11 allows nonprogrammers to design forms interactively, right on the videoscreen, without first drafting the form on paper. It eliminates tedious editing and recompiling of a forms program to see whether the form is satisfactory. Users appreciate the easy-to-learn keypad-operated editor and the HELP facility. Users also like the extensive field protection and validation features that help prevent typing errors.

FMS-11 makes it easy to use the distinctive video attributes of Digital's videoterminals — reverse video, bold, blink, underline, 132-column lines, jump or smooth scrolling, split or reverse screen. In addition, character data types within fields (pictures) are checked on a character-by-character basis. Furthermore, special symbols used for formatting can be embedded within a field without breaking the field into smaller fields. And programs are coded to be completely independent of the forms layout, because form and field names are not bound to the program until execution time.

Programmer benefits include

- Easy, interactive design and maintenance of video forms and application programs.
- Field and record-level I/O calls.
- Reduced memory usage.
- Increased application flexibility.
- Supported languages: BASIC-PLUS-2, COBOL-81, FORTRAN-IV, FORTRAN-77, DIBOL-83, and MACRO-11.
- Supported PDP-11 operating systems: RT-11, RSX-11M, RSX-11M-PLUS, and RSTS/E.

User benefits include

- Keypad-operated editor to speed training and use.
- Field access by name for complete rearrangement of a form without changing the application program.
- Extensive field protection and validation to help prevent errors.
- Online help for every field and every form, reducing documentation and training.

Description

FMS-11 is a set of utilities and subroutines that provide flexible screen formatting for applications written in assembler or high-level languages. A special purpose interactive editor is used to create FMS-11 form definitions for display on the VT52 and VT100 family of terminals (VT100, VT101, VT102, and VT125). The Form Utility provides a means for maintaining disk-resident form libraries, creating listings or object modules of forms descriptions, and listing directories. Application programs control the operator's interaction with the form by subroutine calls to the FMS-11 Form Driver subroutine library.

FMS-11 Forms

An FMS-11 form is a videoscreen image composed of data fields with protection, validation information, and constant background text. The data fields and background text can be highlighted using VT52 and VT100 video attributes such as reverse video, underlining, blinking, and bold characters. Split screen and scrolling capabilities permit users to view more data than can be displayed on the screen at one time.

Figure 21-1 shows a sample of a screen form generated by the Form Editor and demonstrates the use of various display attributes, such as reverse, bold, and underline.

```
Sample of FMS Video Attributes that use the VT100's Video Capabilities

Normal :  This line has only the assigned form-wide attribute.

          (single attribute)       & Reverse        & Bold

Under- :
line      SPRING upward            FLY away         DRIVE like mad

Reverse:  FLOAT gently             DRIFT slowly     SLIDE smoothly

Bold   :  STAND tall               BE strong        LIVE dangerously

Underline  FMS uses the video features of the VT100 to create on any part
& Reverse  of the screen areas of immediate noticeability.  FMS video
& Bold  :  attributes are easy to use and help distinguish different
           sections of a form.
COMMAND:
```

Figure 21-1 ▪ Sample Screen Form

Individual data fields can be display-only or enter-only (noecho). Data fields can be formatted with fill characters, default values, and formatting characters (such as the dash in a phone number), which assist the user but are not visible to the application program. Fields may be right- or left-justified or may use a special fixed-decimal field type to align data properly.

Field validation includes checking each keystroke in a field for the proper data type (for example, alphabetic or numeric). Fields may also be defined as "must enter" or "must complete."

A line of information can be associated with each field, and a chain of HELP screens may be associated with each form. If users need help, they press the HELP key to read a line of useful information about the current field. Subsequent key depressions yield more HELP information.

FMS-11 Programs

A number of components are used to create FMS-11 applications: the Form Editor (FED), the Form Utility (FUT), the Form Driver (FDV), and, for RT-11 only, the Application Run Time Supervisor.

Figure 21-2 illustrates the relationship of FMS-11 components to each other and to an application program.

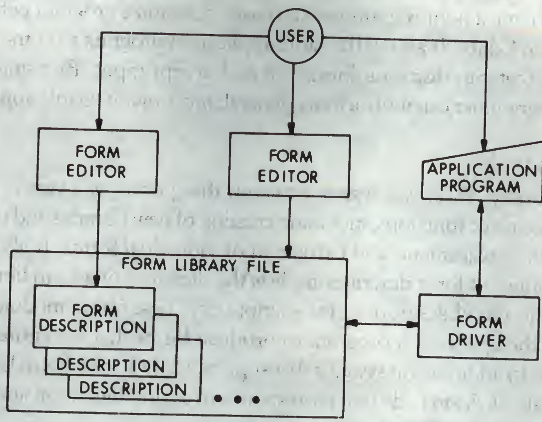


Figure 21-2 ▪ FMS-11 Components

The Form Editor

The Form Editor (FED) is a system program that provides a simple means of entering, modifying, and storing FMS-11 form descriptions. FED lets non-programmers customize existing, general application programs by creating or modifying form descriptions. (This is possible because the form descriptions are independent of the applications that use them). FED is an interactive program that uses many of the special capabilities of the VT52 and VT100 families.

When using FED, a videoscreen always shows the current state of the form that is being edited. Keypad and keyboard functions allow users to specify videodisplay characteristics for either constant text or field characters. Fields are defined on the screen with picture characters similar to those used in COBOL. Short, helpful explanations about individual fields and about each form as a whole can be included as part of the form description.

Fields and forms are accessed by name, rather than by less flexible structures such as row-column coordinates and sizes. Because the relationship between programs and fields is made at execution time, a form can be completely restructured without changing the applications that use the form.

Another feature enhancing program flexibility is *named data*. Named data provides a mechanism for storing constant data (such as file names and range check parameters) in the form description, rather than with the application program. With named data, a nonprogrammer can easily customize program parameters with the Form Editor. It allows the same application program to communicate with a form that can display information and accept input. By using named data, the programmer can write a more general, more maintainable application.

Form Utility

The Form Utility (FUT) is a system program that performs a variety of form library maintenance functions, including creation of new libraries and the insertion, deletion, replacement, and extraction of individual forms. It also creates hardcopy listings of form descriptions, lists the directory of a form library, and produces object modules from form descriptions. These object modules can be linked with the application program to produce forms that are entirely memory-resident. In addition, on systems that support COBOL, the Form Utility can write out Data Division code that corresponds to a form definition and is suitable for copying into a COBOL source program.

Form Driver

The Form Driver (FDV), a reentrant system subroutine, uses the forms created with FED. Under the direction of the calling program, FDV displays forms, performs all screen management a form requires, handles all terminal I/O for application programs, and validates user responses by checking each response against the field and form descriptions. Depending on the needs of the application, programs and forms may interact on either a field-by-field or a whole record basis. FDV may be called from applications written in any of the following languages supported by the operating system—BASIC-PLUS-2, COBOL-81, FORTRAN-77, FORTRAN-IV, DIBOL-83, or MACRO-11. Because the Form Driver calls are virtually identical in all languages, proficiency in using FMS-11 is easily transferable across languages and operating systems, and programs themselves become portable.

Application Run Time Supervisor

Available only with FMS-11/RT-11, the Application Run Time Supervisor (ARTS) controls the interface between form applications and the RT-11 operating system. ARTS runs in the background area of the RT-11 foreground-background monitor, along with the individual application programs, called tasks. When an application is running, each terminal has its own copy of a task. ARTS includes a demand scheduler that handles resources and processing activities so that each terminal runs its task independently.

ARTS allows multiterminal applications to be written in FORTRAN IV or MACRO-11. In multiterminal applications, concurrent tasks can share public files and resident code libraries. Applications include global system tasks not attached to any terminal. Terminal tasks use a system task by sending messages to it and receiving messages from it.

Form applications using ARTS may be either dynamic or static. Dynamic systems allow each terminal to change tasks without affecting the tasks executing at other terminals. Static systems provide fixed relationships between tasks and terminals, locking each terminal into its own individual task. Static systems are most suitable when users do the same work for long periods of time and when that work can be implemented as a small, fixed number of tasks.

Form application systems that use FDV but do not include any of the special ARTS multitask or multiterminal capabilities are also available. The most common use for such a system is debugging and testing tasks as they are being written.

The FMS-11 system generation procedure uses a clear interactive dialogue to select the ARTS features required for a particular application. The hardware on which the application executes may be predefined at ARTS generation time, or it may be specified when ARTS begins to run.

FMS-11 Example

The following code fragment (Figure 21-3) is a sample of FORTRAN application code. FMS Form Driver calls, FGET and FPFT, are used to emulate a call to get all fields, but allow the calling program to validate responses immediately on entry, before proceeding to the next field in the form.

```
      CALL FCLRSH (FORM)                  ! Display the form

      CALL FGET (RESP, TERM, "*")         ! Get first field in form
      CALL FGCF (FIELD)                   ! Get the name of the field
      GOTO 2                              ! Validate response if
                                          ! necessary

1     CALL FGET (RESP, TERM, FIELD) ! Get a field

2     .
      . Validate the user's response.
      . Following validation, the variable "ERRVAL' is zero
      . if the response is valid, non-zero if invalid.
      .
      IF (ERRVAL .NE. 0) GOTO 1           ! Get field again on error
      IF (TERM .EQ. 0) GOTO 10            ! Branch if terminator was
                                          ! "ENTER

      CALL FPFT                           ! Else process field terminator
      CALL FGCF (FIELD)                   ! Get name of field to get
      GOTO 1                              ! Get next field

10    CALL FRETAL (DATA)                  ! Return responses for all
                                          ! fields
      .
      .
      .
```

Figure 21-3 ▪ *Sample Code*

▪ INDENT Simplifies Your Application Program Development

INDENT (Interactive Data Entry) is a forms management and data entry system that greatly simplifies and expedites application program development of programs written on RSTS/E systems in DIBOL, COBOL-81, or BASIC-PLUS-2. INDENT eliminates the many complexities associated with traditional forms management in commercial applications. It benefits both user and developer, allowing more efficient program development and more cost-effective application processing.

INDENT's powerful runtime system is written in reentrant, relocatable MACRO-11 code. It controls all forms execution and communication between the application program and forms jobs running on the system. The runtime system also allows a multiterminal application to engage simultaneously in many tasks with only a single host application program.

INDENT lets you create forms, enter data through these forms, and validate data. Form definitions are created using a text editor. It supports Digital's VT52 and VT100 terminals and uses the following features:

- Reverse video.
- Bold.
- Underline.
- Blink.
- 132-column lines.
- Scroll.
- Split-screen.
- Reverse screen.
- Line drawing character set.

These features produce highly functional, aesthetically pleasing formats.

INDENT offers very flexible screen handling. Forms can be displayed all at once or one field at a time. Fields from different forms can be displayed one after another, while fields from a single form can be displayed in an order different from that in which they occur in the form. Forms can be chained together and portions of a form can be scrolled on the screen. Several forms can also be displayed simultaneously.

INDENT's powerful set of forms directives and host program commands offer more capabilities. Boxes and lines can be quickly defined; tables can include lists of literals or variables; form variables can be defined and accessed by host programs; forms can initialize host programs and host programs can initialize forms; defaults can be set and reset.

Application programming is simple and fast. Changes to host programs, data management, or forms definitions can be implemented independently of one another. Because the INDENT command language is so easy to use, an entry-level programmer can quickly learn to create and modify forms definitions. The field definitions are simple enough so the forms designers don't have to be programmers.

[illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Chapter 22 ▪ Distributed Processing and Networks

▪ Networks Let Your Processors and Terminals Exchange Information

Digital produces powerful computer hardware and software products that permit the linking of computers and terminals into flexible configurations called networks. With networks, you can optimize the efficiency and cost-effectiveness of your data processing operation.

No matter where your processors or terminals are located—around the plant or around the world—they can be connected in ways that allow exchange of information, files, programs, and control, as well as the sharing of peripherals. When networked, small computers can access the powerful capabilities of mainframes; large computers can take advantage of smaller dedicated systems which have been chosen for specific application environments.

With *distributed processing*, computers are placed at the locations where they are needed, whether on the floor of a manufacturing plant, in an accounting department, in a laboratory, or in a home office. As organizations become more complex or develop more sophisticated demands for their computer resources, the ability to network processors and share resources becomes increasingly important. Digital has the distributed processing and networking products to provide customers with these essential capabilities.

Digital's networking architecture offers a broad range of compatible networking options.

- *DECnet* is a family of software and hardware networking products based on the framework of Digital Network Architecture (DNA) for system interconnection. Using DECnet-DOS, Digital systems can be linked to IBM PCs.
- Digital offers a family of protocol emulators and gateways called *Internets*. Internets provide a way for Digital computers and terminals to communicate with computers and terminals built by CDC, IBM, UNIVAC, and several other companies. Users of IBM mainframes, for example, might find it efficient to distribute a number of Digital minicomputers at various locations to do local computing and then link them to the headquarters central computer to give management access to important information quickly and accurately.
- *Packetnet System Interfaces* (PSI) are computer and terminal interfaces to public packet-switched networks. These interfaces allow Digital computers to communicate with those from other manufacturers through a public data network and to other Digital systems.

Each of these series of products will be outlined in detail in this chapter. First, however, some important networking concepts will be introduced.

A *node* is a network entity—an identifiable unit capable of processing, sending, and receiving network information. A node consists of an autonomous software entity, such as an operating system, with processors and other associated hardware that implement DNA specifications. Every node in a DECnet network has a unique numeric address. This address designates the location of a node in a network, just as a street address designates the location of a building in a town.

There are basically two types of nodes — routing and nonrouting. A routing node can receive messages from remote nodes and pass them to nodes other than those linked directly to itself. This sophisticated capability allows for more complex transfer of information around the network, but requires more software "overhead" to manage it. Nonrouting nodes can receive messages from anywhere in the network, but cannot pass them along. For this reason they are also sometimes referred to as "end nodes."

The phrase "point to point" describes the physical configuration of a network. In point-to-point networks, each node must have a physical link to any other node it wants to communicate with. Two point-to-point networks are shown below.

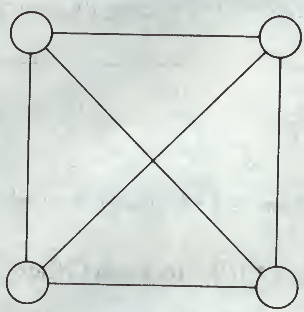


Figure 22-1 ▪ Fully Communicating Point-to-Point Network

The routing network shown in Figure 22-2 provides the same level of communication as does a fully communicating point-to-point network, but with a reduced number of lines/modems needed by using node A as a routing node.

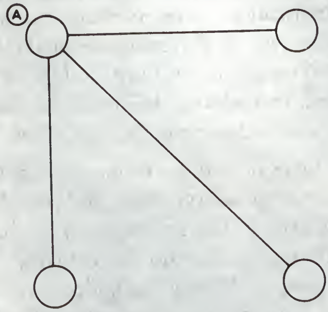


Figure 22-2 ▪ Routing Network

Multidrop connections, however, are more like party telephone lines: several nodes share a single telephone line or local line. Below is an illustration of a multidrop network. It is important to note that one computer is always designated as the master or control node and all other computers on that line are slaves or tributaries and respond to polling by the master node.

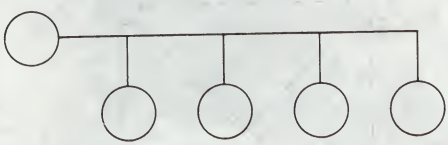


Figure 22-3 ▪ Multidrop Network

▪ DECnet Lets Systems "Talk" to Each Other

DECnet expands the power of Digital computers so that each system is used to its full advantage independently, and, as part of a network, is provided with additional computing benefits. There is a DECnet product to support each of Digital's major operating systems. Capabilities vary for particular DECnet systems, but generally include the following.

Adaptive Routing

Routing ensures that information from a source node is delivered to a specified destination node even if the information has to travel through several intervening nodes. In the event a line fails, DECnet's *adaptive routing* feature finds an available alternative path.

Network Command Terminal

This feature permits a terminal at one network node to have direct logical access to another node. The user's terminal appears to be physically linked to the other system, and the standard network and system utilities of that other system are available for use. Consider, for example, a programmer in one city, say Seattle, who wants to use DATATRIEVE on a huge database stored in a computer in Boston. The Network Command Terminal feature lets that programmer work as if the Boston database were connected directly to the Seattle computer. There is no need to duplicate data and software on multiple systems.

Network Management

Tools for monitoring and controlling network operation are the substance of network management. For day-to-day operations, the architecture includes facilities for tuning parameters, for logging events, and for testing nodes, lines, modems, and communications interfaces.

Loopback testing is a valuable aspect of network management. A network manager can send and receive test messages over individual lines, either between nodes or through other loopback arrangements, and then compare the messages. Utilities are included for a logical series of tests that aid in isolating software and hardware problems.

Access to network performance information allows potential problems to be solved before they degrade performance. If A-to-B traffic increases, the line capacity can be increased.

Task-to-Task Communication

In task-to-task communication, cooperating programs exchange data. These programs can be running under different operating systems; they can be written in different languages. One important benefit of this feature is that the network manager can restrict access to specific programs.

As an example of networked task-to-task communication, think of a manufacturing test/material handling application. A group of PDP-11/44s is testing system components in MACRO and FORTRAN-77 under RSX-11. Once a day, the Operations Manager reviews production schedules. The number of units that have passed final test and are ready for shipment is important information for the COBOL data analysis program running under RSTS/E at headquarters. To the failure analysis program in each RSX-11 system, the running tally it keeps of the number of units which pass the test is relatively insignificant. When the COBOL program goes to get the information for its production scheduling, though, it's there and available.

File Transfer

All DECnet systems support exchange of sequential ASCII or binary files. The DECnet software handles compatibility issues among operating systems by translating the file syntax of the sending node into a common network syntax and then retranslating at the receiving end appropriately for that node.

The transfer of file types other than sequential ASCII and binary may also be supported between particular operating systems. Check with your Digital Network Specialist for details.

As an example of file transfer, think back to the task-to-task example. Since the Operations Manager in that example needed only one record in the FORTRAN program's file, a task-to-task solution made sense. If additional information in that file had also interested the Operations Manager, transferring the whole file might have been preferable.

Remote Resource Access

Sharing such resources as peripheral devices or massive database files is economical as well as convenient. A person at one node can make use of a remote disk storage device or specialized peripheral equipment that makes overhead projection transparencies from word-processor files, just as if they were nearby.

This capability is sometimes also referred to as "remote file (or record) access" when programs access file-structured devices remotely. Figure 22-4 illustrates remote resource access.

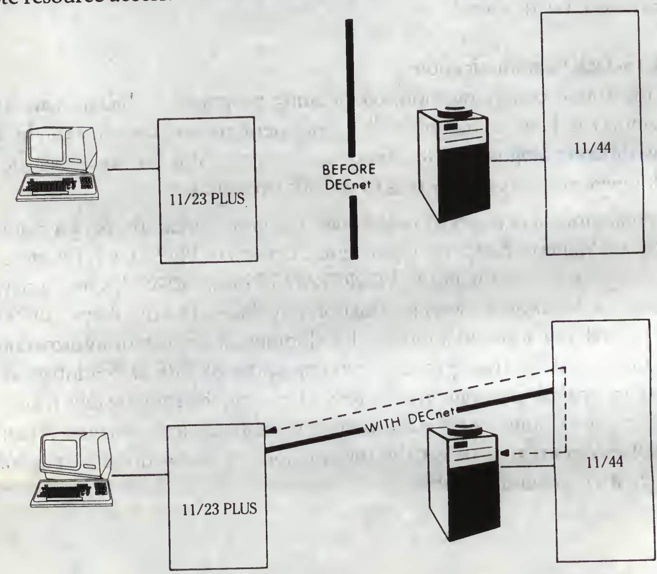


Figure 22-4 ▪ Remote Resource Access

Remote Command File Submission and Execution

A user at a source node requests a destination node to execute a command file. The command file may already be at the destination node, or the source may send it along with the request. This capability turns the user console into a remote job entry terminal for a system that supports batch or indirect command processing. In combination with the file transfer function, it's very powerful.

A small lab system, for example, might acquire data but might not be powerful enough to run the data analysis program. This could be run as a batch job on a larger processor, with the commands submitted from the lab. Later, the file containing the results could be transferred back to the lab for review.

DECnet can also support full batch capability with log file and spooling on base systems that support batch. On RSX-11M-PLUS, for example, the user can generate the node with either command file or batch capability.

Downline Loading

Tasks (programs) or whole software systems can be developed on a node with good peripheral, compiler, and memory support and then be sent to the computer where they will ultimately be used.

Downline task loading is invaluable for final check-out of application programs that have been developed by a central, but geographically separated, applications programming group.

Downline system loading (and its converse, upline system dumping) is particularly useful for small memory-based systems or for systems in hostile environments.

The example in Figure 22-5 shows an RSX-11S system (the only system supported for downline loading) monitoring conditions in a coal mine. It is a likely candidate for downline task or system loading. New applications, once tested, can be downline task loaded. Likewise, the whole system can be regenerated and downline system loaded. The applications or system programmer can do this from a comfortable location with good computer peripheral support. Programs already executing on the memory-only RSX-11S node can be checkpointed to the disk file system of an adjacent node and later restored to main memory of the RSX-11S node. No hardhat and elevator descent are required.

Upline dump of crash information is also supported for memory-only nodes. The dump yields the contents of registers and memory. These may be analyzed on RSX-11M and RSX-11M-PLUS systems by a Network Crash Dump Analyzer.

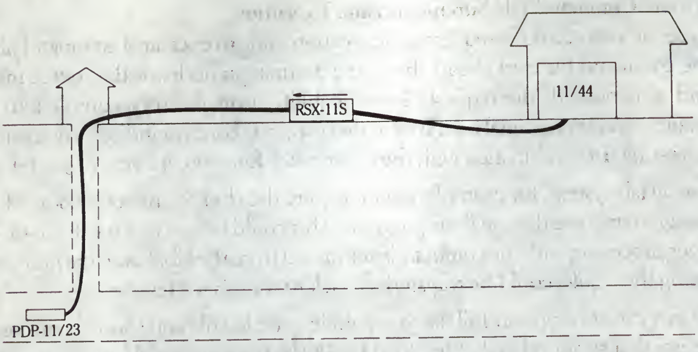


Figure 22-5 ▪ *Downline Loading*

Terminal Communication

The terminal communication facility has many applications. It's a handy way to talk to a person at another terminal or system console. You can request the computer operator at headquarters to mount a magnetic tape with data you need to produce a report remotely. You can tell a remote operator that you're ready to send data and that he should be prepared.

Autoanswer

Once someone dials the number of a destination node, DECnet answers the call, which establishes the communications link, and then attends to the processing requests. Autoanswer is supported by DECnet.

▪ DECnet Phase IV Supports Up to 64K Nodes with Area Routing

DECnet Phase IV is Digital's latest version of network architecture. Phase IV's major enhancement is its support for Ethernet and a series of Ethernet products.

Ethernet is a method for handling high-speed local DECnet communications. Because of rapid advances in microprocessor technology and the widespread acceptance of personal computers, organizations that only recently evolved from centralized mainframe computing to distributed departmental computing can now justify putting a computer in every office and a terminal or personal computer on every desk. Ethernet provides a common communications path by which systems and associated terminals attached by a single connection can communicate with all other attached nodes at speeds comparable to those on the bus that links devices within a single system.

The result is a local area network that can support information exchange within, and among, all levels and departments in your organization. DECnet application programs can be run without modification on Ethernet local area networks. A single Ethernet network can support up to a thousand nodes, although you can start with as few as you wish and add more as they're needed. Ethernet gives you the flexibility to add or remove nodes quickly and easily, without disrupting ongoing communications and without redefining or reconfiguring your entire network.

In order to get an overview of the entire DECnet picture, refer to Table 22-1. It compares the capabilities of all Digital's different DECnet products running on PDP-11 operating systems. Note that the names of DECnet products reflect their compatible operating systems; for example, DECnet/E is the DECnet software for the RSTS/E operating system.

▪ Internets Link Your Mainframe

Internets are a family of protocol emulator products that connect Digital computers with non-Digital systems. If you need a way for Digital computers to communicate and exchange data with computers from other manufacturers, Digital provides mechanisms for interchanging data with IBM, UNIVAC, CDC, and other host processors. Digital's goal is not to provide plug-compatible replacements for terminal subsystems, but instead, to interchange data by using common communications protocols. Emulating a protocol already recognized and supported on another vendor's system is the easiest way to speak to that system.

While Digital's protocol emulator products appear to another vendor's computers to be supported devices, they are, in fact, parts of powerful Digital systems. You get local file systems, many different languages, transaction processing—a wide selection of computing power.

Table 22-1 ▪ Network Comparisons

Capability	DECnet Products							
	-RT	-11M	-11S	-11M PLUS	Micro/ RSX	/E	-IAS	-ULTRIX
Program-to-Program Communications	X	X	X	X	X	X	X	X
Network Virtual Terminal	X	X	X[1]	X	X	X	X	X
File Transfer	X	X		X	X	X	X	X
Command/Batch File Submission and Execution	X[1]	X	X	X	X	X	X	X
Remote Resource Access	X	X	X[1,2]	X	X	X	X	X
Downline System		X		X	X		X	

[1] Requester-only function.

[2] Local system supports unit record equipment only.

Capability	DECnet Products							
	-RT	-11M	-11S	-11M PLUS	Micro/ RSX	/E	-IAS	-ULTRIX
Downline Task Loading		X		X	X		X	
Upline Dumping		X	X	X	X			
Multipoint	X	X	X	X				
X.25		X		X				

Internet Products Summary

Over the past several years, Digital has developed a large number of Internet products to meet customer communications needs. Internet products emulate these protocols:

- DECnet/SNA.
- IBM Remote Batch: 2780, 3780, HASP Workstation.
- IBM Interactive: 3271.
- UNIVAC Remote Batch: UN1004.
- CDC Interactive/Batch: MUX200.

In addition, Digital now offers the DECnet/SNA Gateway, which links Digital and IBM environments, rather than merely providing single-function communications emulation between two computers. Table 22-2 below shows which members of Digital's Internet family are supported by PDP-11 operating systems.

Table 22-2 ▪ Internet Support by Operating System

	2780	3780	IBM RJE/ HASP	3271	SNA	UNIVAC UN1004	CDC MUX200
RT-11	X	X		X			
CTS-300	X	X		X			
RSX-11M-PLUS	X	X	X	X			
RSX-11M	X	X	X	X	X	X	X
RSX-11S	N/A	N/A	N/A	N/A	N/A	N/A	N/A
RSTS/E	X	X		X			
IAS	X		X				X

DECnet/SNA Gateway

The proliferation of networking and distributed processing in the past decade has produced a need for a level of cooperative computing that exceeds the capabilities of single-function communications. To answer that need, Digital developed the DECnet/SNA Gateway, which instead links entire Digital and IBM network environments.

A DECnet/SNA Gateway frees you to expand either the SNA or DECnet side of your total distributed processing operation without jeopardizing your present hardware and software investment. In effect, it extends the reach of a Digital network to include IBM systems connected by an SNA network. This is accomplished by combining three translator functions (remote job entry, 3270 terminal emulation, and applications program interface) and network management software into a small-packaged PDP-11 front-end processor that's attached as a DECnet node.

The gateway lets you access or update an IBM mainframe database, perform remote job entry tasks, and initiate program-to-program communication. The gateway's functions are transparent to end-users, who can perform tasks from Digital workstations or VT100 terminals as if they were integral parts of an SNA network.

SNA Gateway Access is designed to make the full capabilities of SNA Data Flow Control, Transmission Control, and Path Control layers available to user programs residing anywhere in a DECnet network. SNA Gateway Access Protocol messages are exchanged over a logical link to make the facilities of SNA sessions available remotely.

SNA Gateway Access provides the following Gateway Access routines:

- Gateway Management.
- 3270 Terminal Emulation.
- Remote Job Entry (RJE).
- DISOSS Document Exchange (DDXF).
- Distributed Host Command Facility (DHCF).
- Printer Emulation.
- Advanced Program-to-Program Communication/LU6.2 Programming Interface (APPC).
- 3270 Datastream Programming Interface (DS).
- Application Programming Interface (API).

IBM Remote Batch Protocol Emulators (2780/3780, HASP Workstation)
Three Digital remote job entry (RJE) emulators are available for exchanging data with IBM systems: 2780, 3780, and HASP Workstation Protocol Emulators.

The Digital products offer distinct advantages over the IBM products whose protocols they emulate. At the IBM RJE stations, storage is limited to cards for input and to printer or cardpunch for output from the host. The Digital Internet products, since they are integrated into base operating systems, handle all I/O through file systems. Before submitting programs, data, and commands to the host, the Digital user can create and edit files on his or her computer facilities. Output from the mainframe to the Digital system can be spooled to printer or to disk.

The 2780/3780 and HASP are the most commonly implemented communications protocols. They are excellent, low-risk, entry-level products — turnkey packages with standard IBM system software at the IBM end and a straightforward user interface at the Digital end.

In addition to IBM systems, the 2780/3780 and HASP protocol emulators can be compatible with Honeywell, Data General, and other manufacturers' products.

The IBM 2780/3780 is a cardreader, cardpunch, printer, and control unit. It transmits a single data stream. The IBM operator loads a card deck with JCL (Job Control Language) headers and transmits the batch job. Digital's emulator makes use of Digital's mass storage devices. It simulates a card deck with a JCL header in the file.

HASP is sometimes referred as "HASP Workstation" or "Multileaving Workstation." A real HASP Workstation is, functionally, an enhanced 2780. Besides the reader, punch, and printer, it supports a terminal with CRT and keyboard. Through the Digital HASP Protocol Emulator, operators can communicate directly with the IBM mainframe from a local terminal to control and check the status of jobs on the IBM host. This capability is referred to as remote console support.

The HASP Protocol Emulator product also supports multiple I/O streams, the capability of having several devices and/or file transfers active at the same time. With this multileaving capability a short job can be interleaved while a longer job is running.

IBM Programmable Interactive Protocol Emulator (3271)

Digital's 3271 Protocol Emulator provides an interactive, task-to-task link to an IBM mainframe. It provides a mechanism by which an IBM program and a Digital program can communicate.

A real 3271 (3270 is the IBM series number, 3271 the controller product) is a multidrop BISYNC cluster controller with video terminals and printers. The 3271 can transmit and receive data a screenful at a time. It is often used for form-filling applications with CICS, IMS, or user-coded applications or for general purpose timesharing under TSO.

The Digital emulator appears to the IBM system as a cluster controller. To run it, you need two application programs, one for the IBM side and one for the Digital side, to send and receive the data. The IBM system treats the user application program on the Digital side as just another terminal connected to a control unit. The Digital system can coexist with 3270 terminals in a network. IBM customers can make use of their existing terminal-support application.

The Digital user application program has two responsibilities:

- To identify itself to the protocol emulator by terminal unit and control unit number. This step is also referred to as "attaching a pseudodevice."
- To send/receive data in a format acceptable to the IBM program at the other end.

Note also, that, since the 3271 Protocol Emulator emulates a cluster controller, multiple Digital user application programs can send/receive data simultaneously to/from the mainframe on the same physical line. The 3271 also provides multiline support.

UNIVAC Protocol Emulators (UN1004)

UN1004/RSX is a PDP-11-based software package that communicates with UNIVAC 1100 series computers or other computer systems using the UNIVAC 1004 RMS-1 communications protocol. UN1004/RSX communicates with the host using RMS-1 communications protocol supporting ASCII line codes. UN1004/RSX provides for one synchronous communications circuit to a host computer system.

The software oeprates under the RSX-11M operating system and provides remote job entry terminal emulation. The features of the UN1004/RSX protocol emulator include:

- Sending data in 80-column card format and to receive data in line format or card format.
- Controlling data transfers by console commands entered at the emulated terminal. The emulator permits the terminal operator to direct received data to any RSX-11M system-supported device.
- Handling logical terminal emulation separately from a physical system terminal. Logical terminal assignment is provided and released by the terminal operator. The product supports ASCII line communication codes.

Figure 22-6 illustrates a typical UN1004 configuration.

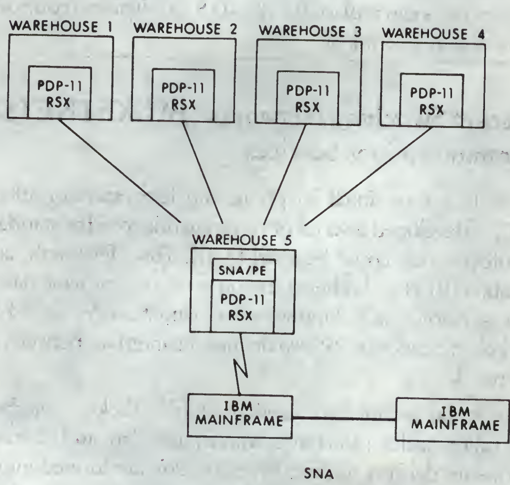


Figure 22-6 ▪ UN1004 Configuration

CDC Protocol Emulator (MUX200)

MUX200/RSX-IAS is a PDP-11-based software package that provides a means of communicating with a CDC-6000- or CDC-CYBER-series host computer. The product may be used to communicate to the host computer either interactively or in remote job entry (RJE) mode. Up to 16 simultaneous users can be connected to the host through MUX200/RSX. However, in many cases the host software or hardware restricts this number. Each of the interactive users may submit jobs to a job queue that MUX200/RSX-IAS uses to schedule transmissions to the host. Output from the host is automatically spooled to a device which has been defined by the system manager.

The protocol used is the CDC mode 4A protocol, which allows speeds up to 9600 baud. Note that the Digital product is warranted for speeds up to 4800 baud.

MUX200/RSX-IAS is not a one-to-one replacement for the CDC-200UT, which uses the same protocol.

MUX200/RSX-IAS has the following features:

- Output received from the host CDC system may be spooled to a lineprinter upon detection of a text string predefined by the user.
- Up to eight RSX-IAS datasets may be specified for transmission to the host in a single command.
- RSX-IAS terminals may be detached for other use while the software package is operating. Data received from the host that are directed to the terminal are saved for printout when the terminal is reattached.
- User-written tasks can replace the RSX-IAS terminal and control the emulator as if the task were a terminal.

▪ Public Packet Switching Networks (PACKETNETS) Provide Data Communications Services

In the 1970s the International Telephone and Telegraph Consultative Committee—CCITT—developed a series of recommendations for standard communications protocols that could be used by the Post, Telegraph, and Telephone Administration (PTT) and other common carriers to provide data communications services. Known as X.25, this recommendation developed for the Public Packet-Switched networks (PPSNs) defines the interface between the computer and the network.

The fundamental technology used in Public Packet-Switched Networks (PPSNs) is called packet switching. With it, user data and control information needed to assure delivery to the correct location are formed into discrete entities/packets. The network dynamically interleaves the packets of many users over shared transmission facilities and routes packets to their destinations. Unlike conventional telephone setups, where the user is charged for both connect time and distance, regardless of the amount of data passed, charges in PPSNs are determined so that the person who uses the line the most pays the most.

Digital's Implementation of the X.25 Interface

Since 1975, Digital has been following the growth of packet-switched networks with great interest and has an ongoing program to provide the X.25 software on its 16-, 32-, and 36-bit computer systems. The X.25 interface has been incorporated into Digital Network Architecture so that Digital-based networks can communicate over both a nationwide PPSN and a private network at the same time. No other computer manufacturer supports this type of approach to networking under multiple computer architectures and operating systems.

X.25 is rapidly becoming the standard international communications protocol, as it finds increasing acceptance among users and computer manufacturers. X.25 allows computers from different manufacturers to work together. With appropriate security validation, any system on the network can send data to any other system on the network. X.25 provides dynamic routing and ensures data integrity, at the same time relieving users of any concern about input and output speeds of the various processors in the network.

Before the advent of packet-switched networks, users required leased or switched lines. These lines are generally not used very efficiently since there are long idle periods between actual data transmissions. Sending data in packets significantly improves the efficiency of the transmission lines, since, by sharing the line among many users, the amount of time the line is idle is reduced. Bandwidth is allocated only when a user is actually transmitting data.

Within the near future, interfaces to PPSNs will be available in:

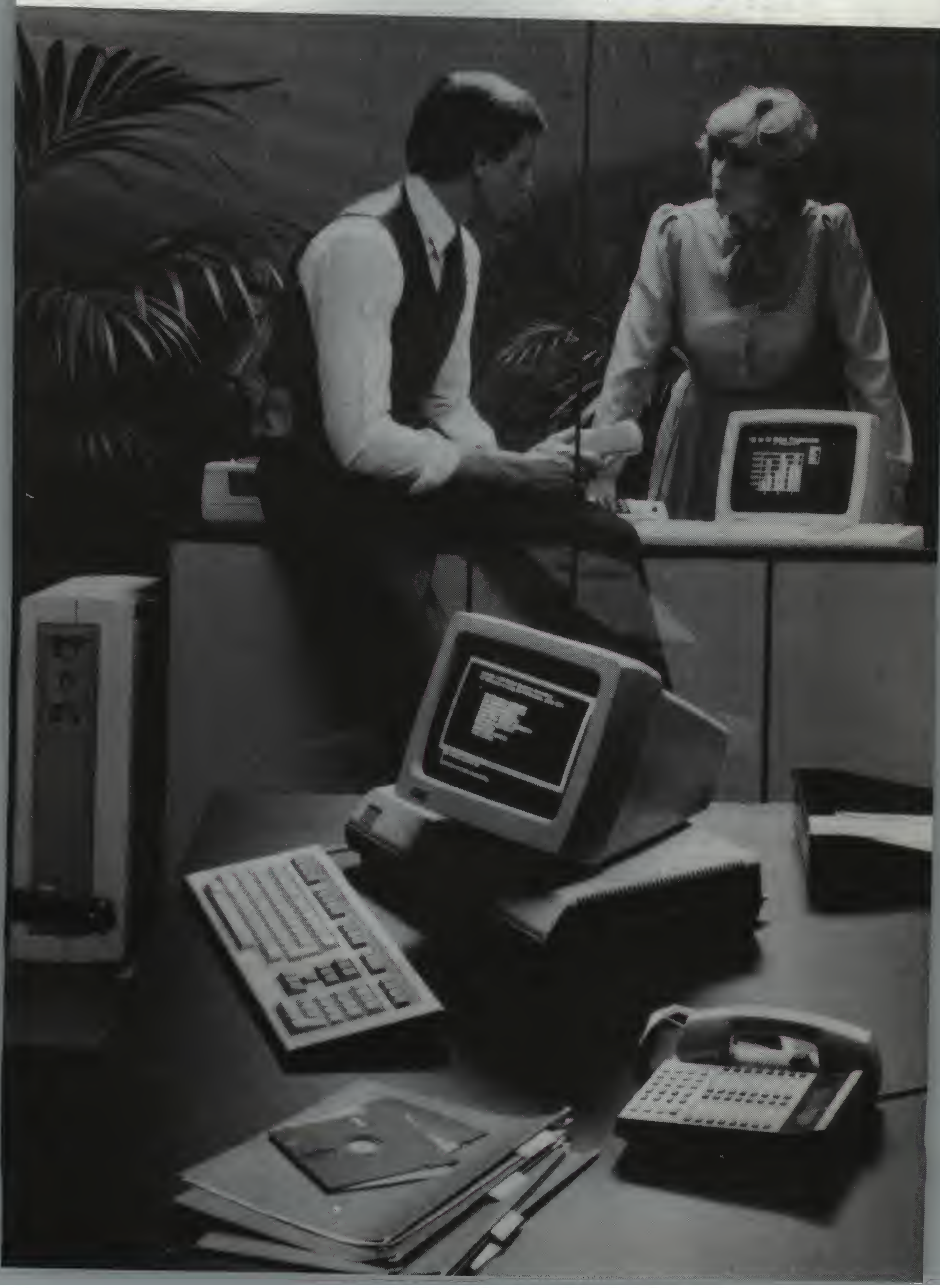
- United States—Telenet.
- United States—Tymnet.
- Canada—Datapac.
- France—Transpac.
- United Kingdom—PSS.
- Holland—DN1 (Not supported by RSX PSI).
- W. Germany—Datex-P.
- Switzerland—Telepac.

As others become operational, they will be selectively supported.

RSX-11 PSI/M

RSX-11 PSI/M allows a suitably configured RSX-11M system to connect to Packet Switching Data Networks (PSDNS) conforming to the CCITT recommendation X.25 (June 1980). Access to RSX-11 PSI/M is supported for RSX-11M user programs written in MACRO-11, FORTRAN IV, and FORTRAN-77. RSX-11 PSI/M supports task-to-task and remote terminal communications via the network.

Chapter 23 ▪ A-to-Z Integrated System



The A-to-Z Integrated System is a user-installable, user-maintainable, small-business system for the MicroPDP-11 family of computers. A-to-Z software is a layered product on the Micro/RSX operating system. The A-to-Z system includes the following features:

- Easy to install by nontechnical computer users.
- Easy to maintain by the novice end user.
- Supports an integrated set of layered products that conform to A-to-Z standards.
- Supports the use of INTERRUPT throughout A-to-Z layered products.
- Provides a set of standards for software developers.
- Provides the ability for growth.

▪ Ease of Installation

The A-to-Z Integrated System will provide the user with a fully configured Micro/RSX system with a user-defined number of A-to-Z user accounts and an A-to-Z Manager account. The A-to-Z Manager is *not* someone who needs to know the operating system or DCL interface, but is merely someone within the organization who is responsible for maintaining user passwords and performing system backup and other simple system functions.

▪ Easy to Maintain by the User

The A-to-Z Manager menu provides all the system management facilities that are required to maintain the A-to-Z system. An example of the A-to-Z Manager menu is shown below. Note that the menu is logically divided so that the user can easily and quickly choose the appropriate option. The major functions required for maintaining the A-to-Z system are

- User Maintenance — This option allows the manager to change user names and passwords as well as to set security codes for selected user menu options.
- Backup and Restore — This option allows the manager to make a complete backup of the A-to-Z system or to selectively backup accounts on the A-to-Z system.
- Install Applications — With this option the manager of the system has the ability to install A-to-Z (or Micro/RSX) layered applications to become part of the user menu.

A-to-Z Integrated Applications

In addition to the A-to-Z Base System that provides the menu interface and system management facilities, Digital provides a set of generic layered applications that are highly integrated with one another. These applications are designed to satisfy the majority of the small-business needs.

A-to-Z Word Processing

This product is compatible with Digital's Gold-Key-style word processing systems found in the DECmate environment. A-to-Z Word Processing provides bolding, underlining, cut and paste, rulers, and User-Defined Keys (UDKs) as well as many other common word processing features. In addition, A-to-Z Word Processing also allows the inclusion of reports and graphs from other components of the A-to-Z system.

A-to-Z Data Inquiry

This product provides the user with the ability to enter and retrieve data from data files and to format the information retrieved in reports, graphs, or list processing data. This product has a command interface and is easy to use.

A-to-Z Electronic Mail

With electronic mail, the A-to-Z user has the ability to work more efficiently in the office by reducing the amount of paperwork in the office. Mail messages can be sent, forwarded, and stored from one user to another. Because A-to-Z Electronic Mail uses A-to-Z Word Processing as its text editor, the user needs only to learn one method for entering text.

Use of the INTERRUPT Feature in A-to-Z Software

A-to-Z software provides the user with the ability to work in react mode by using the INTERRUPT key. A user who is in the middle of a word processing document can simply press the INTERRUPT key and be presented with an interrupt menu. From this menu the user can select a second task to execute. The user may, in fact, wish to interrupt from this second task and execute a third. All A-to-Z generic applications support this interrupt feature.

▪ Provides a Set of Standards for Software Developers

As part of Digital's effort to enable all A-to-Z applications to work in a standard way, Digital has published standards for building A-to-Z applications. For third-party developers who wish to migrate applications to A-to-Z (or start from scratch), there is a wealth of documentation available that explains

- How to use the A-to-Z generic applications in a callable fashion.
- How to create A-to-Z menus.
- How to support the INTERRUPT feature.
- How to create dictionaries that A-to-Z Data Inquiry can use.

▪ A-to-Z Provides the Ability for Growth

With A-to-Z, the small-business user can start small and grow. Because A-to-Z runs on all MicroPDP-11 configurations, a user may wish to start with a PDP-11/23 and later upgrade to a MicroPDP-11/73 and /or MicroPDP-11/83. A-to-Z will move easily from one configuration to the other. Also, because A-to-Z is available by module, a user may wish to start with only one or two layered applications and then add more later. A-to-Z provides this flexibility.

▪ What Is Required to Run A-to-Z?

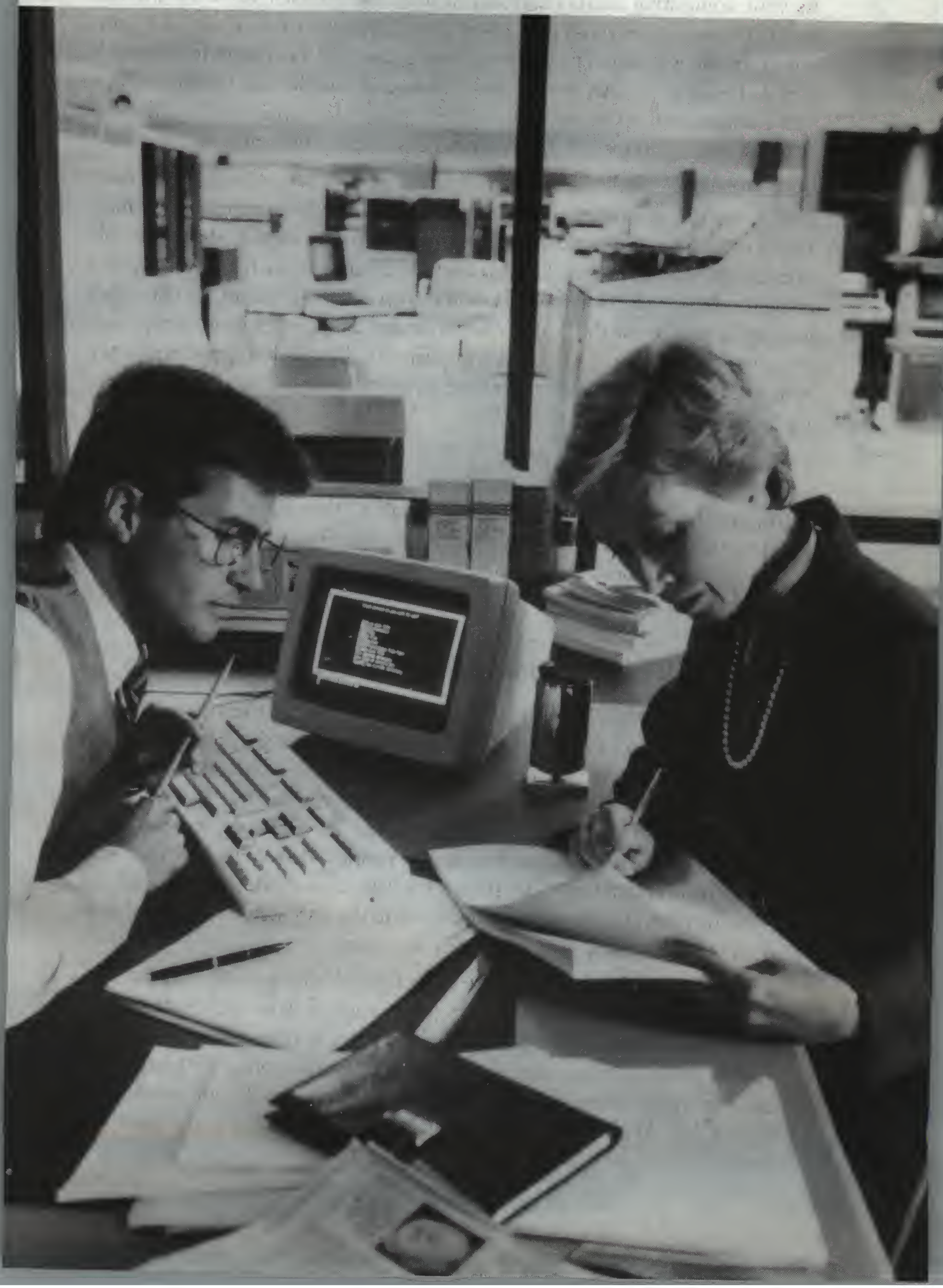
The A-to-Z product is sold in modules so that the user has the flexibility of purchasing only what is needed for the business. The A-to-Z Base System is a prerequisite for running any of the A-to-Z layered products. The products available are

- A-to-Z Base System for MicroPDP-11 license and media/documentation.
- A-to-Z Word Processing for MicroPDP-11 license and media/documentation.
- A-to-Z Data Inquiry for MicroPDP-11 license and media/documentation.
- A-to-Z Business Graphics for MicroPDP-11 license and media/documentation.
- A-to-Z Developer's Kit for MicroPDP-11 license and media documentation.
- A-to-Z Electronic Mail fo MicroPDP-11 license and media/documentation.
- A-to-Z Document Transfer for MicroPDP-11 license and media/documentation.
- A-to-Z Supercomp-Twenty spreadsheet (DCS product).

NOTE

A license for each application must be purchased before a media and documentation kit can be purchased. A-to-Z media kits are availalbe on RX50 diskette or TK50 tape cartridge.

Chapter 24 • PDP-11 and VAX System Coexistence



Coexistence — Digital Makes It Easy to Use Both PDP-11 and VAX Systems

As your computing needs change, you may find yourself wondering if you should choose between using all 16-bit PDP-11 systems or all 32-bit VAX computers. In fact, you have an additional choice: use PDP-11 systems for the work they do best and use VAX systems for the jobs they handle best. Digital — with a firm commitment to compatibility — offers the products you need to allow the different systems to coexist in an efficient computing environment.

If the systems are dedicated to very different work — for example, if you have PDP-11s in manufacturing to control processes and VAX systems in your marketing department for trend analysis — you may require them to share files only occasionally. If they are working on associated projects, such as a VAX system being used to develop software that will run on a PDP-11 system, you probably want to be able to use the same source code and data files. Whatever approach you envision, Digital's PDP-11 and VAX systems will provide a cooperative environment that protects your computing investment as your strategy evolves to meet changing needs.

Discussions on coexistence generally deal with three questions. First, can the systems share data? Second, can the systems run the same programs? And third, are the user interfaces similar enough so that users can move easily between the systems without extensive retraining? The remainder of this chapter answers these questions and provides information on how PDP-11 and VAX systems work together.

Sharing Data Makes Information More Valuable

In virtually every computing environment, at least some of the information created on one system will be useful on another system. Digital provides the data compatibility that meets this need by enabling files to move smoothly from one system to another. This ability to share data gives you the freedom to use the system best suited to various stages of a process. For example, you can set up PDP-11 systems in the laboratory to handle realtime data collection, then send that information to a large VAX system for data analysis. Because the files created on the PDP-11 systems will move easily to the VAX system, you can devote the computers to the type of work each does best.

The basis for this data compatibility is the Record Management System (RMS) that is implemented on RSX and RSTS systems as well as on VAX/VMS systems. RMS provides similar file formats that let you in many cases take a file from one system and use it on another system without conversion. (See Chapter 18, "RMS," for more information about RMS.) RT-11 systems do not use the same RMS, but they do offer utilities that allow you to transfer data to and from VMS systems.

You can transfer data files between the systems either by tape or by your choice of communications or networking software. Additionally, disks created on an RSX system can be transported and read on a VMS system. (Differing disk structures on the other PDP-11 operating systems prevent moving a disk pack from a PDP-11 to a VAX system.)

Digital also offers several communications packages that enable you to do terminal emulation, file transfer, and remote login to and from PDP-11 and VAX systems. PDP-11 and VAX operating systems also support Digital's DECnet networking software. See Chapter 22, "Networking and Data Communications," for more information on the software options available for the PDP-11 operating systems. The *Networking and Communications Buyer's Guide* provides information on the software options for all of Digital's systems.

▪ Application Compatibility Protects Your Software Investment

The PDP-11 and VAX systems protect your existing software investment by enabling you to move applications written for one system to another system. Digital provides this degree of compatibility primarily through our high-level language compilers and secondarily with emulators.

The ease of moving applications between the environments has enabled third-party vendors to offer the same applications for both PDP-11 and VAX systems. With many of the most popular PDP-11 applications also available for VAX/VMS systems, you can create a distributed computing environment in which users work with the same software regardless of which computer system they use.

Compatible Compilers Make It Easy to Move Applications

Because the PDP-11 high-level languages — BASIC-PLUS-2, COBOL-81, FORTRAN-77, and Pascal — are subset compatible with their VAX counterparts, source code moves easily from the PDP-11 to the VAX environment. In many cases, you can recompile the source code on the VMS system and run it. Properly written VMS source code will likewise recompile into executable code for PDP-11 operating system environments. Applications that are written to take advantage of particular hardware or operating system features of one system may require some rewrite to run on the other system.

Digital also provides extensive documentation that explains the differences between the PDP-11 and VAX languages. Some compilers — such as COBOL-81 — even offer a switch that lets you request a report on whether or not the source code is specific to that PDP-11 environment, or whether it will also compile on another system. These tools help you develop applications that will run on both PDP-11 and VAX systems.

When programs are written so that they make direct use of features particular to one system, they do not move as easily to another system. Cross-compilers and translators are available for these situations to help resolve incompatibilities. (For more information on cross-compilers and translators, see the *PDP-11 Software Source Book.*)

Emulators enable you to move object or binary code among the systems. In most cases, MACRO programmers will want to rewrite their code.

Emulators Let You Use Larger Systems to Develop Applications for Smaller Systems

Digital's emulators create the execution environment of one operating system on another operating system. The primary goal of these products is to let you use the resources of a larger system to develop applications that run on the smaller systems — for example, the RTEM-11 emulator runs under RSX and VMS to let you write, compile, and link applications on these larger systems that will run in the smaller RT-11 environment.

Emulators also provide a means for moving an application that was written in MACRO-11 code or that depends on features specific to the PDP-11 or one of its operating systems. The emulator protects your investment in that program by providing a way to move the application to the new environment without having to rewrite it. This approach works best for those applications that are either not used very often or for which performance is not a major consideration. Emulators are also convenient when you are in transition in that they let you use the application on both PDP-11 and VAX systems before making the necessary adjustments for the new environment.

Emulators support the key PDP-11 hardware and software features, but do not support every capability. See a detailed product description for information on the features a particular emulator supports.

▪ Similar User Interfaces Let Users Move Easily Among Environments

If you are setting up an environment in which users will use both systems directly, you need systems that are similar enough for users to move easily between them. The PDP-11 and VAX operating systems provide user interfaces that are similar enough to enable users to move among the systems with ease.

Users familiar with the Digital Command Language (DCL) on RSX, RT, and RSTS operating systems will have little trouble working with the DCL on VMS systems. Because the concepts behind all implementations are the same, users will have a feel for what they can do. The command file processors provide another familiar environment — the command file processors on the RSX and RSTS/E systems are very similar to that on the VMS system. Batch processing on all of the systems is also similar.

At the end-user level, Digital is developing applications that run on the various operating systems to create a consistent user interface. A-to-Z, which creates the same environment whether it is running on a MicroPDP-11 or a MicroVAX system, is an example of this type of product.

Networking and Communications Make It All Transparent

For environments that require many people to work closely together or work on subsets of the same information, Digital provides the DECnet network. The DECnet hardware and software ties both PDP-11 and VAX systems — as well as computers from other vendors — together into a single computing system. DECnet builds on the basic compatibility among the systems to let you create an environment in which users can deal with information distributed across many computers as easily as they can handle information residing on one system.

DECnet runs on RSX, RT, RSTS, and VMS systems to provide — among other capabilities — file transfers, command terminal support, remote file access via applications and, for some systems, X.25 protocol support. All of the high-level languages include calls to the DECnet software to enable you to develop distributed applications that use databases stored anywhere on the network. See Chapter 22, "Networking and Data Communications," for more information on the networking products available for the various PDP-11 operating systems. See the *Networking and Communications Buyer's Guide* for information on networking products available for all of Digital's systems.

Software Options Meet Specific Coexistence Needs

The rest of this chapter describes the software packages Digital offers to help you create an environment in which PDP-11 and VAX systems coexist. For more information on software packages from Digital and from other sources, see the *PDP-11 Software Source Book* and the *VAX Software Source Book*.

VAX-11 RSX. VAX-11 RSX is a software option that creates an RSX execution environment on a VAX/VMS system. VAX-11 RSX makes it possible to use the VAX system to develop applications for RSX-11 systems or to take task images from an RSX-11 operating system and execute them on a VMS system. VAX-11 RSX allows the creation, assembly or compilation, linking, execution, and debugging of programs written for the RSX-11 environment.

RTEM-11. RTEM-11 provides the RT-11 program development environment on RSX-11M, RSX-11M-PLUS, and VMS systems. It takes advantage of the host systems to allow several users to develop RT-11 applications concurrently. RTEM-11 allows the creation, editing, assembling, linking, and debugging of RT-11 applications that can then run on an RT-11 system.

Professional Host Tool Kit. The Professional Host Tool Kit enables you to develop P/OS applications on an RSX-11M-PLUS or VAX/VMS system. Applications are developed on the host computer and transferred, using PRO/Host Communications, to a hard-disk-based Professional system for debugging. Experienced RSX programmers will be productive immediately in this familiar environment.

MicroPower/Pascal. MicroPower/Pascal is an advanced software tool kit that lets you use a VAX/VMS host system to develop Q-bus-based microcomputer applications for PDP-11s. It provides both the host system and target system environment and includes a high-performance Pascal compiler, a modular executive, and a variety of tools to create concurrent realtime applications. The host system creates and builds the software for execution on the target system. Each application is custom-designed for its target system and includes the appropriate set of operating system services. For more information, see Chapter 7, "MicroPower/Pascal."

A-to-Z Integrated System. Digital's A-to-Z Integrated System runs on MicroPDP-11s and MicroVAX systems to provide a consistent, menu-driven user interface for applications. With A-to-Z installed on both Micro/RSX and MicroVMS systems, users work in the same environment regardless of which computer they use. For more information on A-to-Z, see Chapter 23, "A-to-Z System Software."

Appendix A ▪ Customer Services

Like all of Digital's products, PDP-11 software has been designed for reliability and manufactured to strict quality control standards to ensure that design goals are met. Digital's customer services organization is ready to follow up with quality support if it is required. Digital is the complete service vendor and has the products and tools to back its commitment to customer satisfaction.

▪ Field Service

Digital's Field Service is committed to customer satisfaction through quality delivery of a complete range of service products. The service organization complements Digital's hardware offerings and makes a significant contribution to Digital's position as an industry leader.

The Digital Field Service organization includes over 16,000 service engineers who are backed by more than 2,000 administration and support personnel. Backing each service engineer are resources and support programs that help each engineer to meet customer needs effectively. Some of these resources and programs are a computerized logistics network, formalized training programs, an automated call-handling system, remote diagnosis remote support, the site-management guide, and an action planning and problem escalation program.

Onsite Service

Onsite contract services are available for all PDP-11 systems, subject to minimum hardware configurations. These services provide corrective maintenance, preventive maintenance, and all applicable engineering changes to ensure that the microsystems and their options are operational and kept completely up-to-date. In addition to priority service, contractual maintenance allows Digital's customers to budget for their annual maintenance needs. The monthly contract charge covers all travel, labor, and materials. Users have a choice of tailored service agreements. In addition to basic coverage, extended hours are available to customers with critical applications that require special attention.

▪ DECSERVICE

The DECservice agreement is Digital's most comprehensive onsite service product. It provides committed response time including a 4-hour service response if your system is located within 100 miles of a Digital service location. DECservice also provides continuous repairs until the problem is solved, a program of preventive maintenance, installation of the latest engineering changes, and automatic escalation for complex problems. DECservice also offers the customer a choice of coverage hours—up to 24 hours, seven days per week.

- BASIC SERVICE
  The Basic Service agreement is the best alternative for customers whose requirements do not demand a fixed response time to calls for remedial maintenance, or for continuous work to resolve system-down situations outside coverage hours. Basic Service offers economical, yet full-service, coverage. Calls for service receive priority status, second only to DECservice calls. It also provides preventive maintenance, installation of the latest engineering changes, and automatic escalation of complex problems. Ten hours of coverage are during first-shift business hours.

- PER CALL SERVICE
  Per Call Service is a noncontractual, time and materials service. It is available on an onsite and offsite basis. Onsite service is available Monday through Friday during standard business hours, from 8:00 A.M. to 5:00 P.M. Offsite Per Call Service is available through mail-in board replacement and carry-in system repairs.

- SHARED MAINTENANCE SERVICE
  Shared Maintenance Service is a product that combines onsite and offsite services. It is offered to qualified customers who perform their own preventive and remedial maintenance provided that they meet certain prerequisites. The onsite support provided by Digital is similar to a DECservice agreement except that customers, after paying a fixed monthly fee (a percentage of the DECservice maintenance charge), pay only for labor (at the local Per Call rate) and materials as they are required. Shared Maintenance Service features include onsite repairs, committed response, branch telephone support (technical), emergency access to branch logistics, extended coverage (optional), and remote diagnosis (optional where available).

- DECOMPATIBLE SERVICE
  DECompatible Service is a product that provides standard onsite services to selected non-Digital hardware products that are attached to Digital systems. DECompatible Service is provided under DECservice, Basic Service, or Carry-In agreements, depending on the particular device. The level of service and response time under this program is the same quality service as that available for Digital's hardware products.

- RECOVER-ALL SERVICE

Recover-All Service provides full product repair and/or replacement to Digital's hardware products that have been damaged due to accidents or incidents not covered under the other service agreements. Recover-All service expands the customer's service agreement to cover fire, water damage, natural disasters, power failure, sprinkler leakage, and theft. Recover-All Service also provides reimbursement for the cost of movement of equipment to a safe place, returning equipment to the site when safe conditions have been restored, removal of damaged equipment, transportation and installation of replacement equipment, replacement of fire protection chemicals, restoration of damaged Digital system software and customer data from backup disks and tapes, and data processing at a temporary location.

Offsite Services

Customers who do not require onsite services can take advantage of Digital's offsite services that include Digital's Servicenters and DECmailer.

- SERVICENTERS

The Digital Servicenter is a carry-in repair center for Digital's terminals and microsystems. The Servicenter offers low-cost repairs at over 160 convenient locations. At the Servicenter, the same quality service is provided that is given to offsite service calls. The Servicenter guarantees 2-day turnaround time. The customer may select from a variety of service offerings — contract, per call, or parts exchange. All Servicenter service and parts come with a 90-day warranty.

- DECMAILER

DECmailer is a return-to-factory replacement service for customers who maintain their equipment at the module or subassembly level. It provides 5-day turnaround, free return shipping, 90-day warranty on service and parts, 24-hour emergency service, monthly billing, and quarterly activity reports.

▪ Software Services

Digital's Software Services are available to support customers during any phase of their system analysis, software development, or implementation efforts. These services start with the personal attention of a Digital software specialist and continue as long as the customer owns the system.

A Digital software specialist often works with a Digital sales representative to evaluate a prospective user's needs prior to purchase, in order to recommend hardware/software solutions appropriate to the customer's requirements. A full range of services is available to assist customers throughout the planning, implementation, and production phases of their systems.

Software Product Services

For most applications, resources are available to install software. Software support is provided through a variety of Software Product Services that offer customers the opportunity to keep their software up to date and running smoothly through the life of the software.

Software Product Services provide informational, preventive, and remedial service to help customers after the software is installed. These services provide updates to the latest software products, telephone support for remedial questions, and technical publications that contain programming notes and documentation corrections.

- DECSUPPORT SERVICE
  DECsupport is the most comprehensive software product service available. DECsupport includes all of the elements of Basic Service, plus installation of software updates and onsite software support for critical situations.

- BASIC SERVICE
  Basic Service is appropriate for users who do not require onsite support, and includes all of the elements of Self-Maintenance Service, telephone-support service, plus online support via the Digital Software Information Network (DSIN) for usage and remedial software questions. DSIN enables customers to receive software information and solutions to software problems by computer access to a Digital Customer Support Center. Currently, DSIN is available only in the United States and Canada.

- SELF-MAINTENANCE SERVICE
  Self-maintenance Service enables users to maintain their own system software and provides tools that include media and documentation updates, formal software problem-reporting mechanisms, and newsletters and dispatches containing information about new software developments and enhancements.

Startup Service Packages

Software Product Services offers three comprehensive Startup Service Packages for most operation systems. Each package provides media and documentation and one year of service for the operating system and for qualified layered products on that system. Each service package also allows customers to take advantage of several levels of training developed by Digital's Software Services and Educational Services organizations. Training materials and information are available upon purchase of the package.

- STARTUP SERVICE PACKAGE-LEVEL I
  Level I is appropriate for a technical staff requiring minimal training and having the skill to install and support the new system, using the Basic Service telephone-support service to maintain the software at its most current level.

Level I provides

- Basic Service.
- Media and documentation for the operating system and most dependent software purchased concurrently.
- Training.

- STARTUP SERVICE PACKAGE-LEVEL II
Level II is appropriate for a technical staff that can support the new system after Digital has trained the staff, installed the product, and oriented the staff concerning the operation.

Level II provides

- Basic Service
- Media and documentation for the operating system and most dependent software purchased concurrently.
- Installation and DECstart.
- Training.

- STARTUP SERVICE PACKAGE-LEVEL III
Level III includes onsite software support when needed for critical situations and for more comprehensive training.

Level III provides

- DECsupport Service.
- Media and documentation for the operating system and most dependent software purchased concurrently.
- Installation and DECstart Plus.
- Training.

Supplementary Service Options

Software Product Service provides the supplementary options including a Media Update Service, a Documentation Update Service, a Right-to-copy service, an Additional Customer Support Center Contact Service, an Additional Software Dispatch Subscription Service, and a Software Revision Right-to-copy service.

- MEDIA UPDATE SERVICE
  The Media Update Service is a subscription service that provides Software Product services customers with a means of obtaining additional copies of machine-readable media for most operating systems and dependent products. Customers may choose from a variety of distribution media. A prerequisite for the service is that customers have a DECsupport, Basic, or Self-maintenance agreement with Digital.

- DOCUMENTATION UPDATE SERVICE
  The Documentation Update Service supplies service customers with the documentation-only portion of a Software Product Services agreements. The documentation delivered with this service is the portion of the document that was changed or revised since the last release.

- SERVICE RIGHT-TO-COPY
  This option allows customers with a Software Product Service agreement to copy the updates to the product received under agreement onto a single, additional CPU.

- ADDITIONAL CUSTOMER SUPPORT CENTER CONTACT SERVICE
  This service allows customers who have a Basic or DECsupport agreement to add names to the list of people entitled to call the Digital Customer Support Center.

- ADDITIONAL SOFTWARE DISPATCH SUBSCRIPTION SERVICE
  Customers who have a Software Product Services agreement can obtain an additional copy of the dispatches and technical newsletters supplied under the agreement.

- SOFTWARE REVISION RIGHT-TO-COPY
  The Software Revision Right-to-copy option allows customers to copy a single update to the product onto a single, additional CPU.

Installation Service

The purchase of installation as a separate service is appropriate in those instances in which there is no need to purchase a Startup Service Package or a need to have add-on dependent products installed. Installation Service ensures that customers have received all of the proper distribution materials, and that the system generation process for the operating system and/or dependent software products is completed.

DECstart Service

DECstart consists of several levels of fixed-price consulting services with a proven combination of direct assistance, documentation review, discussion, and hands-on experience provided at the customer's site by a Digital Software Specialist. The DECstart services are conducted over a 90-day period to assure mastery of the system. Programmers and system managers are taken step by step through the techniques required to operate their system effectively.

To supplement the DECstart Service, additional services are priced on a time and materials basis. An estimate can be given for any consultation that a customer may be considering. In addition, a Digital Software Specialist can draw up a Customer Support Plan to help the user determine any further areas in which additional services might be beneficial.

▪ Professional Services

Professional Services provides customers with the wide range of onsite services from artificial intelligence to networking.

Network Management Services

Digital's network specialists provide customers with a family of comprehensive network services. These services include Network Planning and Design service, NETstart, OBSERVER, and Network Consulting Services.

▪ THE NETWORK PLANNING AND DESIGN SERVICE

The Network Planning and Design Service provides Digital communications expertise that assists customers in defining or reevaluating network requirements. It develops network designs aimed at meeting customers' business and technical goals.

▪ NETSTART

NETstart, a service designed to help customers use their DECnet networks, is capable of monitoring any DECnet Phase III or IV node. Two NETstart levels are offered—one for customers new to DECnet but familiar with data communications, and one for customers who need assistance in both areas.

- OBSERVER
  OBSERVER, Digital's network monitoring tools and associated services for DECnet networks, is capable of monitoring any DECnet Phase III or IV node. Use of OBSERVER results in a higher level of service for network users, better control over network cost, and more informed planning for network growth and change. Network managers can utilize OBSERVER to identify network problems as they develop and can minimize potential network degradation. Unlike many monitors on the market today, OBSERVER requires no additional software or hardware on the nodes to be monitored.OBSERVER Startup Service is also available, allowing not only quicker, more efficient use of the OBSERVER tool, but also better control over network operations.

- NETWORK CONSULTING SERVICES
  Digital's software specialists assist customers at any stage in their network planning, operation, implementation, or modification. Digital's consultants can provide appropriate assistance, from overall project management to specific advice on a particular problem.

Management Services
These new services bring contemporary planning methodologies and advanced technological approaches together to come up with solutions in the four key areas of change identified by leading experts as critical for business success.

- NETWORK PLANNING AND DESIGN
  Digital's management consulting team will go to customers' sites to help them plan and design a data communications network that supports their business needs, organizational structure, and operational procedures. Customers will learn how to keep costs down, keep up with technology, and continue to expand.

- OFFICE ANALYSIS AND PLANNING
  Digital is ideally positioned to help customers make sound choices for their office environments. First, the professional analyst will study how each department in the customer's organization works. Then, the analyst will determine the technology and applications that will most effectively help them achieve their business goals. The analysts and planners will work closely with all levels of personnel to record goals, objectives, and expectations. They will conduct indepth requirements analysis. When they have collected and evaluated this data, they will present the customers with a formal statement of the findings. In this document, they will recommend how customers can best implement office information technology within their department (or across organizations) to achieve their stated goals.

- CAPACITY PLANNING
  Using specially developed Performance Monitoring and Analysis tools, Digital can help customers restore good service levels and efficiency use, increase their return on investment, and give them information they need to make timely and appropriate decisions on their next equipment purchases.

- ARTIFICIAL INTELLIGENCE
  Digital's experience with Artificial Intelligence technology can offer customers all the traditional Digital support services plus the real advantages of single vendor management. Through Digital's unique multistep methodology, professionals will study the customer's business needs, organizational goals and technological requirements in depth. Then they will help them select the applications with the highest potential payoff and lowest potential risk.

- OFFICE APPLICATION SUPPORT SERVICES
  These services, which provide customized support, feature individualized onsite consulting for office staff, office product usage orientation, office automation transition support, office procedures consulting, and user training on customer applications.

  Digital's customers benefit from an acceleration in productivity gains and improved acceptance of technological changes in the office environment.

- PROJECT SERVICES
  These services provide the customer with the expertise to solve business problems through customized solutions, applications expertise, project management, ongoing support, post-implementation support, and flexible solutions.

- RESIDENT AND ADVISORY SERVICES
  When Digital's customers have an adequate software staff and wish to maintain project management responsibility, Resident and Advisory Services guides customers in effectively using Digital hardware and software. The service features software consulting, short-term staffing, and assistance to personnel using new systems.

- STARTUP SERVICES
  These services are geared toward systems and application management. Their objectives are to optimize return on investment/cost of ownership through increased productivity. Through the use of predefined tasks, customer productivity increases in this short period of time.

▪ Educational Services — Training For Productivity and Performance

Training turns potential into performance and transforms knowledge and initiative into expertise and improved job performance. Productivity is the tangible result of developing a well-trained team equipped to manage information and the tools of technology.

Digital Expertise
To get the maximum return on an investment in Digital products, participate in the most up-to-date, job-relevant training from the people who know Digital products best. For more than 25 years, Digital Equipment Corporation has been providing quality training to executives, computer professionals, and support staff, to help them meet the demands of rapid advances in technology and new job responsibilities.

Digital Equipment Corporation's Educational Services division offers a wide range of training programs to support products of Digital Equipment Corporation, and for the computing public in general. From PDP-11 maintenance to operating systems, layered products, and applications training, Educational services' training programs support the productive use of Digital systems in industry, business, and educational institutions around the world.

Digital Diversity
There are over 500 training offerings geared toward different job functions and levels of expertise, from first-time computer users to experienced system managers and programmers. Training is available in several different delivery options to best fit your requirements.

▪ INSTRUCTOR-LED TRAINING
Lecture/laboratory courses are taught by expert instructors at Digital Training Centers or at an "onsite" location of your choice. These courses offer a technically rich, interactive environment that combines practical knowledge with hands-on practice.

Seminars, conducted by leading computer professionals from Digital Equipment Corporation and the industry at large, focus on state-of-art topics and trends for technical and nontechnical professionals. Seminars are offered worldwide at Digital Training Centers and your "onsite" locations.

- SELF-PACED INSTRUCTION (SPI)

  High-quality SPI materials are designed to meet the needs of those who are unable to attend instructor-led offerings, or those who prefer this alternative educational format. There are several types of self-paced instructional materials available:

  - Computer-based instruction, which features online software programs that make interactive training possible.
  - Text and video-based courses in modules that include workbooks, exercises, tests, and video cassettes.
  - Digital's interactive video informational system (IVIS) that integrates text, audio, and high-resolution graphics with video technology to create a dynamic learning environment.

Customized Training

To fit your training requirements, Educational Services will design training programs and courseware tailored to your own applications and system requirements. Our experts also will help you set up your own inhouse training facility, a Digital Learning Center, fully supported by Digital's consultants and instructors.

When to Invest in Training?

To maximize the performance of a Digital investment, training usually begins before the system arrives and should continue as systems grow and utilization increases.

How to Find Out about Training Opportunities?

To obtain more information about Digital's training programs, contact your local Digital Sales Representative and request a copy of the Educational Services training catalog, the *DIGEST.* You may also obtain a *DIGEST* by sending your requests to

Educational Services Digest
Digital Equipment Corporation
BUO/E55-67
12 Crosby Drive
Bedford, Massachusetts 01730

If you have a specific interest, such as instructor-led training, please let us know. Last, to answer any questions you might have about training, call the Customer Assistance telephone line: (617) 276-4373.

Appendix B ▪ DECUS

DECUS (Digital Equipment Computer Users Society), a worldwide association of customers and employees, provides a forum for the exchange of useful information, new program packages, and other innovations among those who use and supply Digital products.

DECUS membership is free — upon application — to anyone with a bona fide interest in products of Digital Equipment Corporation and their use. Membership carries important benefits and opportunities; among these are access to the program library; membership in local, regional, and national organizations; invitations to symposia dedicated to optimal use of Digital equipment; opportunity to present papers and workshops; and, finally, access to special interest groups dedicated to particular uses, languages, operating systems, and hardware configurations.

The program library maintained by DECUS contains over 1,000 active software packages written and submitted by members and Digital employees, and available to you for the media fee and reproduction cost only. Programs in the library range from enhanced editors and cross-compilers to statistics packages, educational packages, and games. A laboratory user could take advantage of various statistics packages or programs that perform Fourier transforms or least squares fitting. There are programs for circuit analysis, resonance simulation, blood-count evaluation, and stress testing, and scores of others that medical, scientific, or engineering customers could use. Business people can find accounting packages, case studies, and payroll programs among the library's offerings. In addition, of course, there is a wide range of data management, display graphics, and enhanced utility programs available.

Local, regional, and national DECUS organizations give members the opportunity to meet other Digital customers and employees in an informal setting. From the monthly local meeting to the semiannual national symposium, the members can discuss their ideas, can learn what others are doing, and can give Digital feedback necessary for improvement and future development of important products. Often, the national meetings in the various countries also provide the stage for major new product announcements by the company and a showplace for interesting developments in both hardware and software technology. At any meeting, members might describe ideas and programs they have implemented or fine-tuning that has been achieved for a particular application. Members give papers, participate in panel discussions, lead workshops, or conduct demonstrations for the benefit of other members.

Many members derive a particular benefit from joining DECUS Special Interest Groups. Special Interest Groups often meet as subsets of regional and national meetings, or they may meet on their own, to discuss their special field. Here, for example, all RSTS/E users or everyone interested in COBOL can have a chance to get together and discuss topics of mutual interest. At present there are 23 Special Interest Groups (SIGs) in the United States alone. Many of the SIGs contribute to the *SIGs Newsletters* and disseminate valuable technical information to members. The SIGs really are the front-line of mutual help and problem solving.

DECUS publishes the *SIGs Newsletters* monthly, focusing on special interests, technical books that contain the compilation of symposia presentations, and a society newsletter quarterly.

Digital provides DECUS with administrative personnel and office space around the world, but the organization is run by its members, who act as speakers for conferences, planners for meetings, editorial and production talent for newsletters and minutes, and the inventors of the ideas and new programs necessary to keep the library up-to-date. Belonging to DECUS is a valuable adjunct to using Digital equipment on both the program exchange and the information exchange fronts.

For further information about DECUS, contact

DECUS
219 Boston Post Road (BPO2)
Marlboro, MA 01752

Appendix C ▪ PDP-11 Software Source Book

PDP-11 computers are the most popular microcomputer and minicomputer family of all time. That popularity is reflected in the fact that the PDP-11 has the greatest number of proven applications available to any micro or mini family in the industry. The fourth edition of the *PDP-11 Software Source Book* covers a collection of over 2,400 of these application packages — not software under development, but software that is available now and already at work in businesses and organizations.

Applications mentioned include those sold by Digital, as well as products available through DECUS (Digital Equipment Computer Users Society) and independent third-party vendors.

The *PDP-11 Software Source Book* consists of two volumes. Volume 1 describes applications software—everything from accounting, agriculture, and arts to service industries, transportation, and utilities. For each entry, the book gives you

- The name of the software product.
- The operating system on which it runs.
- Keywords that define the product, its function, and its application.
- A brief description.
- The price.
- The contact—a person or company—that can provide more information on the product.

Volume 2 describes systems software — topics such as Communications, Language Processors, and Operating Systems — and also contains extensive cross-reference guides. Cross-reference guides are available by keyword and by operating system. A product name list, a vendor name list, and a detailed index complete the second volume and give the users a choice of ways to find the product they need.

The *PDP-11 Software Source Book* is free and can be ordered by writing to

Digital Equipment Corporation
Printing and Circulation Services
Literature Inquiry Fulfillment
10 Forbes Road, NRO3/M1
Northboro, Massachusetts 01532-2597
Ask for Order Code: ED-27333-41.

Appendix D ▪ Documentation

Digital offers several levels of documentation describing PDP-11 software and hardware. These manuals are updated periodically to include new developments and new products.

The hardware user documentation, software tutorial documentation, and reference manuals that accompany the delivery of a PDP-11 computer system offer the most detailed levels of information.

There are also several books published commercially that discuss the PDP-11 family. If you have a specific documentation need, contact a Digital sales representative who will guide you to the appropriate literature.

The following lists contain the titles and associated order numbers of Digital literature that apply to the PDP-11 family.

Title	**Order Number**
PDP-11 Architecture Handbook	EB-23657-18
Terminals and Printers Handbook	EB-26291-56
ULTRIX Software Guidebook	EJ-26153-20
PDP-11 Software Sourcebook—Fifth Edition (Volumes 1 and 2)	EB-27333-41
PDP-11 UNIBUS Processor Handbook	EB-26077-41
Microcomputer Products Handbook	EB-26078-41
Supermicrosystems Handbook	EB-27113-41
PDP-11 Systems and Options Catalog	ED-27983-41
Networks and Communications Buyer's Guide	ED-28252-49
Networks Handbook	EB-26013-42

Appendix E ▪ Commonly Used Abbreviations

A	Amperes
A/D	Analog/digital
ACP	Ancillary Control Processor
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
AST	Asynchronous System Trap
ATL	Active Task List
BAC	BASIC Compiled Program File
BAK	Back-up file
BAS	BASIC source program file
BASIC	Beginner's All-purpose Symbolic Instruction Code
BAT	Batch file
BI	Batch input device
BP	Batch pseudodevice
CAI	Computer-Assisted Instruction
CBL	COBOL source program
CCITT	Comité Consultatif International Télégraphique et Téléphonique (International Telephone and Telegraph Consultative Committee)
CIL	Core Image Library
CL	Console Log device
CLI	Command Language Interpreter
CMD	Command file
CMI	Computer Managed Instruction
CO	Console Output device
COB	COBOL source program
COBOL	Common Business Oriented Language
COMTEX	Communications-Oriented Multiple Terminal Executive

CPU	Central Processing Unit
CR	Card Reader
CRC	Cyclic Redundancy Check
CREF	Cross-Reference
CRT	Cathode Ray Tube
CSECT	Control Section
CSI	Command String Interpreter
CT	Cassette Tape
CTRL	Control key
CUSPs	Commonly Used System Programs
DAT	Data file
DBMS	Database Management System
DDCMP	Digital Data Communications Message Protocol
DEC	Digital Equipment Corporation
DIR	Directory File
DMP	Dump File
DMS	Data Management System
DNA	Digital Network Architecture
DOS	Disk Operating System
DP	Data Processing
DPB	Directive Parameter Block
DSM	Digital Standard MUMPS
DSW	Directive Status Word
DT	DECtape
EAE	Extended Arithmetic Element
EDI	Editor utility
EDIT	Editor utility
EDT	Digital standard editor
EIA	Electronics Industry Association
EIS	Extended Instruction Set
EMT	Emulator Trap
EOD	End of Data

EOF	End of File
EOJ	End of Job
EOL	End of Line
EOM	End of Medium
FA	Formatted ASCII
FB	Formatted Binary
F/B	Foreground/Background
FCP	File Control Primitives
FCS	File Control Services
FDB	File Data Block
FILEX	File Exchange utility
FIS	Floating Instruction Set
FLX	File Exchange utility
FNB	Filename Block
FOR	FORTRAN source program
FORTRAN	Formula Translator
FPP	Floating-point Processor
FSR	File Storage Region
FTN	FORTRAN source program
F4P	FORTRAN IV-PLUS source program
G	Giga (one billion)
GT	Graphics Terminal
HASP	Houston Automatic Spooling Program
Hz	hertz
IAS	Interactive Application System
ID	Identification code
I/O	Input/Output
IOT	Input/Output Trap
IOX	I/O Executive
ISR	Interrupt Service Routine
JMP	Jump
JSR	Jump to Subroutine

K	1,024 decimal (2^{10})
KB	Keyboard
KBL	Keyboard Listener
KCT	Kilo-Core Tick
LBR	Librarian
LDA	Load module
LED	Light Emitting Diode
LIB	Library file
LIBR	Librarian
LICIL	Linked Core Image Library
LIS	Listing file
LP	Line printer
LST	Listing file
LUN	Logical Unit Number
u	micro (mu—one millionth)
m	milli (one thousandth), or meters
M	mega ($1{,}024^2$)
MAC	MACRO source program
MAP	Load map
MCR	Monitor Console Routine
MFD	Master File Directory
MO	Message Output device
MST	Macro Symbol Table
MT	Magnetic Tape
MTBF	Mean Time Between Failures
n	nano (one billionth)
NCP	Network Control Processor
NPR	Nonprocessor Request
OBJ	Object Module
ODL	Overlay Description Language
ODT	Online Debugging Technique
OEM	Original Equipment Manufacturer

OTL	Online Task Loader
OTS	Object Time System
PC	Program Counter
PDF	Processor-Defined Function
PDS	Program Development System
PDP	Programmed Data Processor
PIC	Position Independent Code
PIP	Peripheral Interchange Program
PP	Papertape Punch
PR	Papertape Reader
PSECT	Program Section
PST	Permanent Symbol Table
PTT	Post, Telegraph, and Telephone Administration
PUD	Physical Unit Directory
QIO	Queue I/O
ROM	Read-only Memory
RSTS/E	Resource-sharing Timesharing System/Extended
RSX-11	Realtime Resource Sharing Executive
RT-11	Realtime Foreground/Background System
RTS	Runtime System
RWED	Read, Write, Extend, and Delete
SAV	Saved File or System Image File
SCI	System Control Interface
SCOM	System Communication Area
SGA	Shareable Global Area
SIP	System Image Preservation
SIPP	Save Image Patch Program
SP	Stack Pointer
SPC	Small Peripheral Controller
SPR	Software Performance Report
SST	Synchronous System Trap
SY	System device

SYS	System file
SYSGEN	System Generation
TCP	Timesharing Control Primitives
TI	Terminal Interface
TKB	Task Builder
TKTN	Task Termination Notice
TMP	Temporary file
TSK	Task Image file
TT	Terminal device
TTY	Terminal device
UA	Unformatted ASCII
UB	Unformatted Binary
UFD	User File Directory
UIC	User Identification Code
USR	User Service Routine
UST	User Symbol Table
V	Volts
VDT	Video Display Terminal
VT50	DECscope video display terminal
VT52	Tabletop alphanumeric video display terminal
VT100	High-performance videoterminal
VT200	Expanded features of the high-performance VT100 videoterminal
XOR	Exclusive OR

Appendix F ▪ ASCII Codes

▪ Control Characters

CHAR	OCTAL	BINARY
NUL	000	0000000
SOH	001	0000001
STX	002	0000010
ETX	003	0000011
EOT	004	0000100
ENQ	005	0000101
ACK	006	0000110
BEL	007	0000111
BS	010	0001000
HT	011	0001001
LF	012	0001010
VT	013	0001011
FF	014	0001100
CR	015	0001101
SO	016	0001110
SI	017	0001111
DLE	020	0010000
DC1	021	0010001
DC2	022	0010010
DC3	023	0010011
DC4	024	0010100
NAK	025	0010101
SYN	026	0010110
ETB	027	0010111
CAN	030	0011000
EM	031	0011001
SUB	032	0011010
ESC	033	0011011
FS	034	0011100
GS	035	0011101
RS	036	0011110
US	037	0011111
DEL	177	1111111

Control Character Key

NUL = All zeros
SOH = Start of heading
STX = Start of text
ETX = End of text
EOT = End of transmission
ENQ = Enquiry
ACK = Acknowledgement
BEL = Bell or attention signal
BS = Back space
HT = Horizontal tabulation
LF = Line feed
VT = Vertical tabulation
FF = Form Feed
CR = Carriage return
SO = Shift out
SI = Shift in
DLE = Data link escape
DC 1 = Device control 1
DC 2 = Device control 2
DC 3 = Device control 3
DC 4 = Device control 4
NAK = Negative acknowledgement
SYN = Synchronous/idle
ETB = End of transmitted block
CAN = Cancel (error in data)
EM = End of medium
SUB = Start of special sequence
ESC = Escape
FS = Information file separator
GS = Information group separator
RS = Information record separator
US = Information unit separator
DEL = Delete

▪ Printable Characters

CHAR	OCTAL	BINARY
SP	040	0100000
!	041	0100001
"	042	0100010
#	043	0100011
$	044	0100100
%	045	0100101
&	046	0100110
'	047	0100111
(	050	0101000
)	051	0101001
*	052	0101010
+	053	0101011
,	054	0101100
-	055	0101101
.	056	0101110
/	057	0101111
0	060	0110000
1	061	0110001
2	062	0110010
3	063	0110011
4	064	0110100
5	065	0110101
6	066	0110110
7	067	0110111
8	070	0111000
9	071	0111001
:	072	0111010
;	073	0111011
<	074	0111100
=	075	0111101
>	076	0111110
?	077	0111111
@	100	1000000

▪ Printable Characters

CHAR	OCTAL	BINARY
A	101	1000001
B	102	1000010
C	103	1000011
D	104	1000100
E	105	1000101
F	106	1000110
G	107	1000111
H	110	1001000
I	111	1001001
J	112	1001010
K	113	1001011
L	114	1001100
M	115	1001101
N	116	1001110
O	117	1001111
P	120	1010000
Q	121	1010001
R	122	1010010
S	123	1010011
T	124	1010100
U	125	1010101
V	126	1010110
W	127	1010111
X	130	1011000
Y	131	1011001
Z	132	1011010

CHAR	OCTAL	BINARY
a	141	1100001
b	142	1100010
c	143	1100011
d	144	1100100
e	145	1100101
f	146	1100110
g	147	1100111
h	150	1101000
i	151	1101001
j	152	1101010
k	153	1101011
l	154	1101100
m	155	1101101
n	156	1101110
o	157	1101111
p	160	1110000
q	161	1110001
r	162	1110010
s	163	1110011
t	164	1110100
u	165	1110101
v	166	1110110
w	167	1110111
x	170	1111000
y	171	1111001
z	172	1111010

Glossary

absolute address: A binary number that is assigned as the address of a physical memory storage location.

absolute loader: A stand-alone program that, when in memory, enables the user to load into memory data in absolute binary format.

access privileges: Attributes of a file that specify the class of users allowed to access the file.

account number: A discrete code used to identify a system user. It normally consists of two numbers, separated by a comma, called the project number and programmer number or the group number and member number. *See also* **user identification code.**

active task list: A priority-ordered list of active tasks used normally in an event-driven multiprogrammed system to determine the order in which tasks receive control of the CPU.

address: 1. A name, label, or number that identifies a register, a location in storage, or any other data source or destination. 2. The part of an instruction that specifies the location of an operand of that instruction.

adjacent node: A node removed from the local node by a single physical line.

algorithm: A prescribed set of well-defined rules or processes for the solution of a problem; a program.

alphanumeric: Referring either to the entire set of 128 ASCII characters or the subset of ASCII characters that includes the 26 alphabetic characters and the ten numeric characters.

ancillary peripherals: In the DSM-11 system, peripherals not under control of the database supervisor.

ANSI: American National Standards Institute.

append: To add information to the end of an existing file.

application program: A program that performs a task for a particular end user's needs. Generally, an application program is any program written on a program development operating system that is not part of the basic operating system.

argument: 1. A variable or constant that is given in the call of a subroutine as information to it. 2. A variable upon whose value the value of a function or other operation depends. 3. The known reference factor necessary to find an item in a table or array (e.g., an index).

array: An ordered arrangement of subscripted variables.

ASCII: The American Standard Code for Information Interchange, consisting of 128 7-bit binary codes for uppercase and lowercase letters, numbers, punctuation, and special communication control characters.

assemble: To translate from a symbolic program to a binary program by substituting binary operation codes for symbolic operation codes and absolute or relocatable codes and absolute or relocatable addresses for symbolic addresses.

assembler: A program that translates symbolic source code ("assembly level language") into machine instructions by replacing symbolic operation codes with binary operation codes and symbolic addresses with absolute or relocatable addresses.

assembler directives: The mnemonics used in an assembly language source program that are recognized by the assembler as commands to control and direct the assembly process.

assembly language: A symbolic programming language that can normally be translated directly into machine language instructions and is, therefore, specific to a given computing system.

assembly listing: A listing produced by an assembler that shows the symbolic code written by a programmer next to a representation of the actual machine instructions generated.

assigning a device: Putting an I/O device under control of a particular user's job either for the duration of the job or until the user relinquishes control. *See also* **attach.**

asynchronous: A mode of operation in which an operation is started by a signal that the operation on which it depends is completed. When referring to hardware devices, it is the method in which each character is sent with its own synchronizing information. The hardware operations are scheduled by ready and done signals rather than by time intervals. In addition, it implies that a second operation can begin before the first operation is completed.

asynchronous system trap: A system condition that occurs as the result of an external event such as completion of an I/O request. On occurrence of the significant event, control passes to an AST service routine.

asynchronous transmission: Time intervals between transmitted characters may be of unequal length. Transmission is controlled by start and stop elements at the beginning and end of each character. Also called Start-Stop transmission.

attach: To dedicate a physical device unit for exclusive use by the task requesting attachment. *See also* **assigning a device.**

background processing: The automatic execution of a low-priority computer program when higher-priority programs are not using the system resources.

backup file: A copy of a file created for protection in case the primary file is unintentionally destroyed.

bad block: A defective block on a storage medium that produces a hardware error when attempting to read or write data in that block.

base address: An address used as the basis for computing the value of some other relative address.

base segment: The always-memory-resident portion of a program that uses overlays. *See also* **root segment.**

batch processing: A method of scheduling programs in which programs are accumulated and fed to the computer for execution with no programmer interaction.

batch stream: The collection of commands and data interpreted by a batch processor that directs batch processing.

baud: 1. A unit of signaling speed equal to the number of signal events per second. 2. For asynchronous transmissions, the unit of modulation rate corresponding to one unit interval per second. If, for example, the length of the unit's interval is 25 milliseconds, the modulation rate is 40 baud. Baud is frequently, though erroneously, used as a synonym for bits per second.

binary: The number system with a radix of two.

binary code: A code that uses two distinct characters, usually the numbers 0 and 1.

binary loader: *See* **absolute loader.**

bit: A binary digit.

bit map: A table describing the state of each member of a related set. A bit map is most often used to describe the allocation of storage space. Each bit in the table indicates whether a particular block in the storage medium is occupied or free.

block: 1. A group of specified size of physically adjacent words or bytes. A block size is peculiar to a device. 2. The smallest system-addressable segment on a mass-storage device in reference to I/O.

Boolean valued expression: An expression that, when evaluated, produces either "true" or "false" as a result.

bootstrap: 1. A technique or device designed to bring itself into a desired state by its own action. 2. To cause an operating system to load itself and prepare to run.

bootstrap loader: A routine whose first instructions are sufficient to load the remainder of itself into memory from an input device and normally start a complex system of programs.

bottom address: The lowest memory address in which a program is loaded.

bps: Bits per second. A commonly used measure for data transfer rate. (Other notation is bit/sec.)

breakpoint: A location at which program operation is suspended in order to examine partial results. A preset point in a program at which control passes to a debugging routine.

buffer: A storage area used to hold temporarily information being transferred between two devices or between a device and memory. A buffer is often a special register or a designated area of memory.

bug: An instruction or sequence of instructions in a program that causes unexpected and undesired results.

bus: One or more conductors used for transmitting signals or power from one or more sources to one or more destinations, but usually with many connections.

byte: The smallest memory-addressable unit of information in a PDP-11 system. A byte is equivalent to eight bits.

call: To transfer control to a specified routine.

calling sequence: A specified arrangement of instructions and data necessary to pass parameters and control to a given subroutine.

carriage return key: The key on a terminal keyboard most often used in PDP-11 systems to terminate input lines.

CCITT: Comité Consultatif International Télégraphie et Téléphonie, a committee that sets international communications standards.

Central Processing Unit (CPU) or Central Processor: That part of a computing system containing the arithmetic and logical units, instruction control unit, timing generators, and memory and I/O interfaces.

character: A single letter, numeral, or symbol used to represent information.

checkpoint: A point in a program or routine at which the job and system status are recorded so that the job can later be restarted.

checksum: A number used for checking the validity of data transfers.

clear: 1. To erase the contents of a storage location by replacing the contents with zeros or spaces. 2. In binary code, to set to zero.

clock: A time-keeping or frequency-measuring device within a computing system.

code: A system of symbols and rules used for representing information.

coding: Writing instructions for a computer using symbols meaningful to the computer itself, or to an assember, compiler, or other language processor.

collate: To combine items from two or more ordered sets into one set having an order not necessarily the same as any of the original sets.

command or command name: A word, mnemonic, or character that, by virtue of its syntax in a line of input, causes a predefined operation to be performed by a computer system.

command language: The vocabulary used by a program or set of programs that directs the computer system to perform predefined operations.

Command Language Interpreter: The program that translates a predefined set of commands into instructions that a computer system can interpret.

command string: A line of input to a computer system that generally includes a command, one or more file specifications, and optional qualifiers.

Command String Interpreter: A special program or routine that accepts a line of ASCII string input and interprets the string as input and output file specifications with recognized qualifiers.

common: A section in memory set aside for common use by many separate programs or modules.

compatibility: The ability of an instruction, source language, or peripheral device to be used on more than one computer.

compile: To translate a source (symbolic) program into a binary-coded program. In addition to translating the source language, appropriate subroutines may be selected from a subroutine library. Linkage is supplied, and everything is output in binary code along with the main program.

compiler: A program that translates a higher-level source language into a language suitable for a particular machine.

completion routine: A routine that is called at the completion of an operation.

compute bound: A state of program execution in which all operations depend on the activity of the central processor, for example, when a large number of calculations is being performed. *Contrast with* **I/O bound.**

computer operator: A person who performs standard system operations such as adjusting system operation parameters at the system console, loading a tape transport, placing cards in a card reader, and removing listings from the lineprinter.

concatenate: To combine several files into one file, or several strings of characters into one string, by appending each file or string one after the other.

conditional assembly: The assembly of parts of a symbolic program only when certain conditions are met.

configuration: A particular selection of hardware devices or software routines or programs that function together.

consecutive access: The method of data access characterized by the sequential nature of the I/O device involved. A card reader is an example of a consecutive-access device. Each card must be read after the preceding one, and no distinction is made between logical sets of data in or among the cards in the input hopper.

console: In a central processor, the set of switches and display lights used by an operator or programmer to determine the status and control the operation of the computer.

console terminal: A keyboard terminal that acts as the primary interface between the computer operator and the computer system and is used to initiate and direct overall system operation through software running on the computer.

constant: A value that remains the same throughout a distinct operation. *Compare with* **variable.**

context switching: The switching between one mode of execution and other, involving the saving of key registers and other memory areas prior to switching between jobs, and restoring them when switching back. A common example of context switching is the temporary suspension of a user program so that the monitor or executive can execute an operation.

contiguous file: A file consisting of physically adjacent blocks on a mass-storage device.

control character: A character whose purpose is to control an action rather than to pass data to a program. An ASCII control character has an octal code between 0 and 37. It is typed by holding down the CTRL key on a terminal keyboard while striking a character key.

control section: A named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated separately without destroying the logic of the program.

core memory: The most common form of main memory storage used by the central processing unit, in which binary data is represented by the switching polarity of magnetic cores.

core common: *See* **common.**

crash: A hardware crash is the complete failure of a particular device, sometimes affecting the operation of an entire computer system. A software crash is the complete failure of an operating system characterized by some failure in the system's protection mechanisms.

create: To open, write data to, and close a file for the first time.

cross-reference listing or table: A printed listing that identifies all references in a program to each specific label in a program. A list of all or a subset of symbols used in a source program and statements where they are defined or used.

CTRL/C (C): The control character issued from a terminal that is most commonly used to return the operator to communication with the system-level program. In most PDP-11 systems, it is typed on the terminal keyboard to gain the attention of the operating system before commencing the login procedure, or to terminate the currently executing program and return to communication with the monitor. In some cases, it simply issues a call to the console listener or console service routine without interrupting current program execution.

CTRL/U (U): The control character issued from a terminal that tells the program currently accepting input to ignore the characters entered on the line up to the point at which CTRL/U was typed.

CTRL/Z (Z): The control character used in RSX-11 systems to terminate the system program currently waiting for input from the terminal. It is essentially an end-of-file character.

database: A collection of interrelated data items organized by a consistent scheme that allows one or more applications to process the items without regard to physical storage locations.

database management system: A scheme used to create, maintain, and reference a database.

debug: To detect, locate, and correct coding or logic errors in a computer program.

DECnet: A family of hardware/software products that create distributed networks from Digital computers and their interconnecting datalinks.

DECtape: A convenient, pocket-sized reel of magnetic tape developed by Digital for extremely reliable data storage and random access.

default: The value of an argument, operand, or field assumed by a program if a specific assignment is not supplied by the user.

delimiter: A character that separates, terminates, or organizes elements of a character string, statement, or program.

detach a device: Free an attached physical device unit for use by tasks other than the one that attached it.

device: A hardware unit such as an I/O peripheral, e.g., magnetic tape drive or card reader. Also often used synonymously with volume.

device controller: A hardware unit that electronically supervises one or more of the same type of devices. It acts as the link between the CPU and the I/O devices.

device driver: A program that controls the physical hardware activities on a peripheral device. The device driver is generally the device-dependent interface between a device and the common, device-independent I/O code in an operating system.

device handler: A program that drives or services an I/O device. A device handler is similar to a device driver, but provides more control and interfacing functions than a device driver.

device independence: The ability to request I/O operations without regard for the characteristics of specific types of I/O devices.

device name: A unique name that identifies each device unit on a system. It usually consists of a two-character device mnemonic followed by an optional device unit number and a colon. For example, the common device name for DECtape drive unit one is "DT1:".

device unit: One of a set of similar peripheral devices; e.g., disk unit 0, DECtape unit 1. Also used synonymously with volume.

diagnostic: Pertaining to the detection and isolation of malfunctions or mistakes.

dialup line: A communications circuit that is established by a switched circuit connection.

Digital Network Architecture (DNA): The common network architecture of DECnet.

digital transmission: Transmission of data characters that are coded into discrete separate pulses or signal levels.

direct access: *See* **random access.**

direct mode: The mode of DSM-11 system operation that enables the programmer to enter commands and/or functions for immediate execution, and to create or modify steps of a user's program.

directive: A type of executive request issued by a program that provides a facility inherent in the hardware that is controlled and organized by the operating system. *See also* **programmed request.**

directory: A table that contains the names of and pointers to files on a mass-storage device.

directory device: A mass-storage retrieval device, such as disk or DECtape, that contains a directory of the files stored on the device.

disk: 1. A mass-storage device. Basic unit is an electromagnetic platter on which data is magnetically recorded. Features random access and faster access time than magnetic tape. 2. The platter itself.

double-buffered I/O: An input or output operation that uses two buffers to transfer data. While one buffer is being used by the program, the other buffer is being read from or written to by an I/O device.

downline-load: The process by which one node in a computer network transfers an entire system image or a program (task) image to another node and causes it to be executed.

dump: To copy the contents of all or part of core memory, usually onto an external storage medium. Also, the copy so produced.

EBCDIC: Extended Binary Coded Decimal Interchange Code. A binary code used on the IBM System/360 and System/370 as well as other manufacturers' systems that have a need to be compatible with IBM.

echo: The printing by an I/O device, such as teletype or CRT, of characters typed by the programmer.

editor: A program that interacts with the programmer to enter new programs into the computer and edit them as well as modify existing programs. Editors are language-independent and will edit anything in alphanumeric representation.

emulator: A hardware device that permits a program written for a specific computer to be run on a different type of computer system.

executive: The controlling program or set of routines in an operating system. The executive coordinates all activities in the system including I/O supervision, resource allocation, program execution, and operator communication. *See also* **monitor.**

executive mode: A central processor mode characterized by the lack of memory protection and relocation by the normal execution of all defined instruction codes.

exponentiation: A mathematical operation denoting increases in the base number by a factor previously selected.

expression: A combination of operands and operators that can be evaluated by a computing system.

external storage: A storage medium other than main memory, for example, paper tape, magnetic tape, or disks.

field: 1. One or more characters treated as a unit. 2. A specified area of a record used for a single type of data.

file: A logical collection of data treated as a unit. It occupies one or more blocks on a mass-storage device such as disk, DECtape, or magtape. A file can be referenced by a logical name.

file gap: A fixed length of blank tape separating files on a magnetic tape volume.

file name: The alphanumeric character string assigned by a user to identify a file, and that can be read by both an operating system and a user. A file name identifies a unique member of a group of files which 1. has the same file name extension and version number (if any), 2. is located on the same volume, and 3. belongs in the same User File Directory (if any). A file name has a fixed maximum length that is system-dependent (generally six or nine characters).

file specification: A name that uniquely identifies a file maintained in any operating system. A file specification generally consists of at least three components: a device name identifying the volume on which the file is stored, a file name, and a file type. In addition, depending on the system, a file specification can include a User File Directory name or UIC, and a version number.

file structure: A method of recording and cataloging files on mass-storage media.

file-structured device: A device on which data is organized into files. The device usually contains a directory of the files stored on the device.

file type: The alphanumeric character string assigned to a file either by an operating system or a user, and that can be read by both the operating system and the user. System-recognizable file types are used to identify files having the same format or type (e.g., FORTRAN source files might have the file type .FOR in their file specification). A file type follows the file name and is separated from it by a period. A file name extension has a fixed maximum length that is system-dependent (generally three characters, excluding the preceding period).

flag: A variable or register used to record the status of a program or device. In the latter case, it is sometimes called a device flag.

floating-point numeric: A floating-point number that, if stored in four words, is approximately in the range 10^{-38} to 10^{38}.

foreground: 1. The area in memory designated for use by a high-priority program. 2. The program, set of programs, or functions that gain the use of machine facilities immediately upon request.

format: The arrangement of the elements constituting any field, record, file, or volume.

formatted ASCII: Refers to a mode in which data is transferred. A file containing formatted ASCII data is generally transferred as strings of 7-bit ASCII characters (bit eight is zero) terminated by a linefeed, formfeed or vertical tab. Characters, such as NULL, RUBOUT, and TAB, may be interpreted specially.

formatted binary: Refers to a mode in which data is transferred. Formatted binary is used to transfer checksummed binary data (8-bit characters) in blocks. Formatting characters are start-of-block indicators, byte count, and checksum values.

formatted device: A volume that has been prepared for use on a system under program control.

frame: That part of the packet carrying data required by the link control protocol and defining the leading and trailing ends of the bit stream.

full-duplex: A line that can transmit data in both directions simultaneously. A full-duplex line allows a node to send and receive data at the same time.

fully connected network: A network in which each node is directly connected with every other node.

function: An algorithm accessible by name and contained in the system software. It performs commonly used operations, such as the square root calculation function.

generation number: *See* **version number.**

global: A value defined in one program module and used in others. Globals are often referred to as entry points in the module in which they are defined, and externals in the other modules that use them. Also, in the DSM-11 system, a global array.

global array: A data file stored in the common DSM-11 database. Global arrays constitute an external system of symbolically referenced arrays.

global variable: A global variable in the DSM-11 system is a subscripted variable that forms a part (or node) of a global array.

half-duplex: The line that can transmit data in either direction, but only in one direction at any given time. In other words, the line cannot be used to send and receive data simultaneously.

handler: *See* **device handler.**

hardcopy: A printed copy on some kind of paper, generally in readable form, such as listings and other documents.

hardware: The physical equipment components of a computer system.

HASP: Houston Automatic Spooling Program. An IBM 360/370 OS software front end that performs job spooling and controls communications between local and remote processors and Remote Job Entry (RJE) stations.

higher-level language: A programming language whose statements are translated into more than one machine language instruction. Examples are BASIC, FORTRAN, and COBOL.

host: A computer connected to a network and implementing its protocols in such a way that its computing power is accessible through the network.

host node: A node that provide services for another node. For example, the host node supplies program image files for a downline-load.

idle time: That part of uptime in which no job could run because all jobs are halted or waiting for some external action such as I/O.

image mode: A mode of data transfer in which each byte of data is transferred without any interpretation or data changes.

impure code: The code that is modified during the course of a program's execution, e.g., data tables.

incremental compiler: A compiler that immediately translates each source statement into an internal format, ready for execution.

indirect file or indirect command file: A file containing commands that are processed sequentially, yet which could have been entered interactively at a terminal.

indirect mode: The mode of DSM-11 system operation in which steps of a stored program can be executed. In this mode, neither commands nor functions can be entered at the terminal, nor can programs be created or modified.

indirect reference: A feature of the MUMPS language that permits the symbolic representation of an argument or argument list in a command by a string variable. In operation, the string value of the variable is taken as the argument or argument list for the command. The indirection symbol, a back-arrow (←) or underscore (_) must precede the variable reference.

initialize: To set counters, switches, or addresses to starting values at prescribed points in the execution of a program, particularly in preparation for reexecution of a sequence of code. To format a volume in a particular file-structured format in preparation for use by an operating system.

input: 1. Data to be processed. 2. The process of transferring data to memory from a mass-storage device or from other peripheral devices that read data from other media.

instruction: One unit of machine language, usually corresponding to one line of assembly language, that tells the computer what elementary operation to do next.

interactive: A technique of user/system communication in which the operating system immediately acknowledges and acts upon requests entered by the user at a terminal. *Compare with* **batch processing.**

interface: A shared boundary, for example, the wires and perhaps other electronics connecting two subsystems.

Internet: A network linking Digital computers to non-Digital computers.

interpreter: A computer program that translates and executes each source language statement before translating and executing the next statement.

interrupt: A signal that, when activated, causes a transfer of control to a specific location in memory, thereby breaking the normal flow of control of the routine being executed. An interrupt is normally caused by an external event such as a done condition in a peripheral. It is distinguished from a trap that is caused by the execution of a processor instruction.

interrupt service routine: The routine entered when an external interrupt occurs.

interrupt vector address: A unique address that points to two consecutive memory locations containing the start address of the interrupt service routine and priority at which the interrupt is to be serviced.

I/O bound: A state of program execution in which all operations depends on the activity of an I/O device. For example, when a program is waiting for input from a terminal. *Compare* **compute bound.**

I/O page: That portion of memory in which specific storage locations are associated directly with I/O devices.

I/O rundown: A process that delays the availability of a partition until all transfers to and from that partition have been stopped or have been allowed to complete. I/O rundown is invoked when a task is terminated and has outstanding transfers pending to or from its partition.

job: A group of data and control statements that does a unit of work, e.g., a program and all its related subroutines, data and control statements; also, a batch control file.

journaling: The parallel writing of updated records to a second medium in addition to the original file.

K: 1. An abbreviation for the prefix kilo, i.e., 1,000 in decimal notation. 2. In the computer field, two to the tenth power, which is 1,024 in decimal notation. Hence, a 4K memory has 4,096 words.

keyboard monitor: A program that provides and supervises communication between the user at the system console and an operating system.

latency: 1. The time from initiation of a transfer operation to the beginning of actual transfer; i.e., verification plus search time. 2. The delay while waiting for a rotating memory to reach a given location.

leader: A blank section of tape at the beginning of a reel of magnetic tape or at the beginning of paper tape.

leased-line: A line reserved for the exclusive use of a leasing customer without interchange switching arrangements. Also called a private line.

library: 1. A file containing one or more relocatable binary modules that are routines that can be incorporated into other programs. 2. A class of MUMPS programs listed in the system program directory and available to all users of the system.

line: 1. A string of characters terminated with a vertical tab, formfeed, or linefeed. 2. The network management component that provides a distinct physical data path.

linked file: A file whose blocks are joined together by references (a link word or pointer embedded in the block) rather than consecutive location.

linker: A program that combines many relocatable object modules into an executable program module. It satisfies global references and combines control sections.

linking loader: A program that provides automatic loading, relocation, and linking of compiler and assembler generated object modules.

listing: The hardcopy generated by a lineprinter.

literal: An element of a programming language that permits the explicit representation of character strings in expressions and command and function elements. In most languages, a literal is enclosed in either single or double quotes to denote that the enclosed string is to be taken "literally" and not evaluated.

load: 1. To store a program or data into memory. 2. To mount a tape on a device such that the read point is at the beginning of the tape. 3. To place a removable disk in a disk drive and start the drive.

load image file: A program that can be executed in a stand-alone environment without the aid of relocation.

load map: A table produced by a linker that provides information about a load module's characteristics, e.g., the transfer address and the low and high limits of the relocatable code.

load module: A program in a format ready for loading and executing.

local node: A frame of reference; the node at which the user is physically located.

local variable: In the DSM-11 system, a variable that is stored only in the partition in which a program is executed (as opposed to a global variable).

location: An address in storage or memory in which a unit of data or an instruction can be stored.

log in: To identify oneself to an operating system as a legitimate user of the system and gain access to its services.

log out or log off: To sign off a system.

logical block: An arbitrarily defined, fixed number of contiguous bytes that is used as the standard I/O transfer unit throughout an operating system. For example, the commonly used logical block in PDP-11 systems is 512 bytes long. An I/O device is treated as if its block length is 512 bytes, although the device may have an actual (physical) block length that is not 512 bytes. Logical blocks on a device are numbered from block 0 consecutively up to the last block on the volume. A logical block is synonymous with a physical block on any device that has 512-byte physical blocks. *See also* **logical record, physical block, physical record, virtual block.**

logical device name: An alphanumeric name assigned by the user to represent a physical device. The name can then be used synonymously with the physical device name in all references to the device. Logical device names are used in device-independent systems to enable a program to refer to a logical device name that can be assigned to a physical device at runtime.

logical link: A carrier of a single stream of full-duplex traffic between two user-level processes.

logical record: A logical unit of data within a file whose length is defined by the user and whose contents have significance to the user. A group of related fields treated as a unit.

logical unit number: A number associated with a physical device unit during a task's I/O operations. Each task in the system can establish its own correspondence between logical unit numbers and physical device units.

machine language: The language, peculiar to each kind of computer, that that computer understands. It is a binary code that contains an operation code to tell the computer what to do, and an address to tell the computer on which data to perform the operation.

macro: Directions for expanding abbreviated text. A boilerplate that generates a known set of instructions, data, or symbols. A macro is used to eliminate the need to write a set of instructions that are used repeatedly. For example, an assembly language macroinstruction enables the programmer to request the assembler to generate a predefined set of machine instructions.

main memory: The set of storage locations connected directly to the Central Processing Unit. Also called (generically) core memory.

main program: The module of a program that contains the instructions at which program execution begins. Normally, the main program exercises primary control over the operations performed and calls subroutines or subprograms to perform specific functions.

mapped system: A system that uses the hardware memory management unit to relocate virtual memory addresses.

mass storage: Pertaining to a device that can store large amounts of data readily accessible to the Central Processing Unit; for example, disk, DECtape, and magnetic tape.

master file directory: The system-maintained file on a volume that contains the names and addresses of all the files stored on the volume.

matrix: A rectangular array of elements. A table can be considered a matrix.

memory: Any form of data storage, including main memory and mass storage, in which data can be read and written. In the strict sense, memory refers to main memory.

memory image: A replication of the contents of a portion of memory.

memory mapping: A mode of computer operation in which the high-order bits of a virtual address are replaced by an alternate value, providing dynamic relocatability of programs.

memory protection: A scheme for preventing read and/or write access to certain areas of memory.

modulo: A mathematical operation that yields the remainder function of division. Thus 39 modulo 6 equals 3.

monitor: The master control program that observes, supervises, controls, or verifies the operation of a computer system. The collection of routines that controls the operation of user and system programs, schedules operations, allocates resources, performs I/O, etc.

monitor command: An instruction issued directly to a monitor from a user.

monitor console: The system control terminal.

Monitor Console Routine (MCR): The executive routine that allows the user to communicate with the system using an online terminal device. MCR accepts and interprets commands typed on the terminal keyboard and calls appropriate routines to execute the specified requests.

mount a device or volume: To associate a physical mass-storage medium logically with a physical device unit. To place a volume on a physical mass-storage drive unit; for example, place a DECtape on a DECtape drive and put the drive online.

multiplexing: Use of one communications line circuit for two or more simultaneous data paths.

multipoint: A communication line (circuit) with three or more communicating devices on it (terminals or computers). Use of this type of line normally requires a polling technique with an address for each device. Also called multidrop.

multiprocessing: Simultaneous execution of two or more programs by two or more processors.

multiprogramming: A processing method in which more than one task is in an executable state at any one time.

naked syntax: A feature of the MUMPS language, providing an abbreviated method for accessing global variables, that controls the disk access time. The node reference includes only subscript(s) for the element; the global variable name is assumed from the last global reference in which a name was explicitly stated.

network: A configuration of two or more computers linked to share information and resources. A computer having the capacity to participate in a network is called a node.

node: 1. A dynamically allocated set of bytes from a node pool used for system communication and control in an RSX-11/IAS system. 2. An element of a global array in a DSM-11 system (also called a global variable). 3. A network management component consisting of a system that supports network software.

noncontiguous file: A file whose blocks are not physically contiguous on the volume.

nonfile-structured device: A device, such as paper tape, lineprinter, or terminal, in which data is not referenced as a file.

nonrouting (end) node: A nonrouting node can send packets to other nodes in the network, but it cannot forward packets or route them through itself. It can be adjacent to one other node only; therefore, it is always an end node in a Phase III configuration.

null modem: A device that interfaces between a local peripheral that normally requires a modem, and the computer near it that expects to drive a modem to interface to that device; an imitation modem in both directions.

object code: Relocatable machine language code.

object module: The primary output of an assembler or compiler; it can be linked with other object modules and loaded into memory as a runnable program. The object module is composed of the relocatable machine language code, relocation information, and the corresponding symbol table defining the use of symbols within the module.

object program: The relocatable binary program that is the output of a compiler or assembler.

Object Time System: The collection of modules that is called by compiled code in order to perform various utility or supervisory operations. For example, an Object Time System usually includes I/O and trap handling routines.

octal: Pertaining to the base eight number system.

offline: Pertaining to equipment or devices not under direct control of the Central Processing Unit.

offset: The difference between a base location and the location of an element related to the base location. The number of locations relative to the base of an array, string, or block.

online: Pertaining to equipment or devices directly connected and under control of the Central Processing Unit.

operating system: The collection of programs, including a monitor or executive and system programs, that organizes a central processor and peripheral devices into a working unit for the development and execution of application programs.

output: 1. The results of processing data. 2. The process of transferring data from memory to a mass-storage device or from memory to a copying device such as a lineprinter or paper tape punch. 3. The process of moving information from a mass-storage device to a copying device. 4. The peripheral device receiving the information described above.

overlay description language: The set of instructions interpreted by a linker that defines the overlay structure of a task.

overlay segment: A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments.

overlay structure: A task overlay system consisting of a root segment and optionally one or more overlay segments.

p-section (program section): A section of memory that is a unit of the total task allocation. A source program is translated into object modules that consist of p-sections with attributes describing access, allocation, relocatability, etc.

pack: 1. To compress data in storage by using an algorithm for its storage and retrieval. 2. A removable disk.

packet: The unit of data switched through a Packet Switching Service, normally a user data field accompanied by a header carrying destination and other information and enclosed in a frame, possibly shared with other packets.

packet switching: A data transmission process utilizing addressed packets whereby a channel is occupied only for the duration of transmission of the packet.

parity bit: A binary digit appended to a group of bits to make the sum of all the bits always odd (odd parity) or always even (even parity). Used to verify data storage.

parse: To break a command string into its elemental components for the purpose of interpretation.

part number: In the MUMPS language, the integer portion of a program step that is used to refer collectively to all steps having a common integer base.

partition: A contiguous area of memory within which tasks are loaded and executed.

patch: To modify a program by changing the binary code rather than the source code.

peripheral: Any device distinct from the central processor that can provide input to or accept output from the computer.

Phase II node: A node that runs a Phase II implementation of DECnet and, therefore, does not support routing. It can send packets only to adjacent nodes and it cannot forward packets it receives on to other nodes in the network. It can be adjacent to one or more full-routing nodes and/or to other Phase II nodes. Logically, it is an end node within a Phase III configuration.

Phase III node: A node that runs under a Phase III implementation of DECnet and supports routing as either a full-routing or nonrouting (end) node. *See also* **routing node.**

physical address space: The set of memory locations where information can actually be stored for program execution. Virtual memory addresses can be mapped, relocated, or translated to produce a final memory address that is sent to hardware memory units. The final memory address is the physical address.

physical block: A physical record on a mass-storage device.

physical device: An I/O or peripheral storage device connected to or associated with a central processor.

physical record: The largest unit of data that the read/write hardware of an I/O device can transmit or receive in a single I/O operation. The length of a physical record is device-dependent. For example, a punched card can be considered the physical record for a card reader; it is 80 bytes long. The physical record for an RK11 disk is a block; it is 512 bytes long.

position-independent code: Code that can execute properly wherever it is loaded in memory, without modification or relinking. Generally, this code uses addressing modes that form an effective memory address relative to the central processor's Program Counter (PC).

priority: A number associated with a task that determines the preference its requests for service receive from the executive relative to other tasks requesting service.

privilege: A characteristic of a user or program that determines what kinds of operations that user or program can perform. In general, a privileged user or program is allowed to perform operations normally considered to be the domain of the monitor or executive or that can affect system operation as a whole.

program development: The process of writing, entering, translating, and debugging source programs.

programmed requests: An instruction (available only to programs) that is used to invoke a monitor service.

programmer access code: The system identification code that enables a user to gain access to a DSM-11 system in direct mode to create, modify, and execute programs.

project-programmer number: *See* **account number.**

protocol: A formal set of conventions governing the format and relative timing of message exchange between two communicating processes.

pseudodevice: A logical entity treated as an I//O device by the user or the system, but that is not actually any particular physical device.

PTT: Abbreviation for the post, telegraph, and telephone adminstrations that act as common carriers for telecommunications in many European and other countries.

public disk structure: The disk volume or set of volumes that are used as a general storage pool available to any users having quotas on the public structure.

pure code: Code that is never modified during execution. It is possible to let many users share the same copy of a program that is written as pure code.

qualifier: A parameter specified in a command string that modifies some other parameter. *See also* **switch.**

queue: Any list of items; for example, items waiting to be scheduled or waiting to be processed according to system- or user-assigned priorities.

Radix-50: A storage format in which three ASCII characters are packed into a 16-bit word.

random access: Access method in which the next location from which data is to be obtained is not dependent on the location of the previously obtained data.

read: To transfer information from a peripheral device into core memory or into a register in the CPU.

realtime processing: Computation performed while a related or controlled physical activity is occurring, so that the results of the computation can be used in guiding the process.

record: A collection of adjacent data items treated as a unit. *See* **logical record, physical record.**

record gap: An area between two consecutive records.

recursive: Pertaining to a process that is inherently repetitive. The result of each iteration of the process usually depends on the result of the previous iteration.

reentrant: The property of a program that enables it to be interrupted at any point by another program, and then resumed from the point where it was interrupted.

relocatable: Describes a routine, module, or segment whose address constants can be modified to compensate for a change in origin.

remote batch terminal: A combination of hardware items (usually a card reader, communication interface and a printer) and a communication link that allows the transmission of a series of records (the job) to a host computer. The host processes the records as a job stream and transmits the results of the job back to the lineprinter. Communiation is typically at the file level.

remote node: A frame of reference; any node other than the one at which the user is located in the network. *Compare* **local node**.

resident: Pertaining to data or instructions that are normally permanently located in main memory.

resource sharing: The joint use of resources available on a network by a number of dispersed users.

restart address: The address at which a program can be restarted. It is normally the address of the code required to initialize variables, counters, etc.

root segment: The segment of an overlay tree that, once loaded, remains resident in memory during the execution of a task.

routine: A set of instructions arranged in proper sequence to cause the computer to perform a desired task.

routing node: A full-routing node can forward packets to other nodes in the network and can be adjacent to all other types of nodes.

secondary storage: Mass storage other than main memory.

segment: 1. That part of a long program that may reside in core at any one time. 2. To divide a program into segments or to store part of a program on a mass-storage device to be brought into memory as needed.

sentinel file: The last file on a cassette tape that represents the logical end-of-tape.

sequential access: A data access method in which records or files are read one after another in the order in which they appear in the file or volume.

shareable program: A (reentrant) program that can be used by several users at the same time.

significant event: An event or condition that indicates a change in system status in an event-driven system. A significant event is declared, for example, when an I/O operation completes. A declaration of a significant event indicates that the executive should review the eligibility of task execution because the event might unblock the execution of a higher-priority task. The following are considered to be significant events: I/O queuing, I/O request completion, a task request, a scheduled task execution, a mark time expiration, a task exit.

single-user access: The status of a volume that allows only one user to access the file structure of a volume.

single-stream batch: A method of batch processing in which only one stream of batch commands is processed.

sliver: A 32-word section of memory.

source language: The system of symbols and syntax, easily understood by people, which is used to describe a procedure that a computer can execute.

sparse array: Refers to the method of storage allocation used in MUMPS for local and global arrays in which space is allocated only as variables are explicitly defined (unlike some other languages that require dimension or size statements for preallocation of storage).

spooling: The technique by which output to low-speed devices (e.g., lineprinters) is placed into queues on faster devices (e.g., disks) to await transmission to the slower devices.

statement: An expression or instruction in a source language.

step number: A number in the range 0.01 to 327.67 used to identify each line of a MUMPS program.

string: A connected sequence of entities such as a line of characters.

subscript: A numeric valued expression that is appended to a variable name to identify specific elements of an array. Subscripts are enclosed in parentheses. Multiple subscripts must be separated by commas. For example, a two-level subscript might be (2,5).

swapping: The process of copying areas of memory to mass storage and back in order to use the memory for more than one purpose. Data is swapped out when a copy of the data in memory is placed on a mass-storage device; data is swapped in when a copy on a mass-storage device is loaded in memory.

swapping device: A mass-storage device especially suited for swapping because of its fast transfer rate.

switch: An element of a command or command string that enables the user to choose among several options associated with the command. In PDP-11 software systems, a switch element consists of a slash character (/) followed by the switch name and, optionally, a colon and a parameter. For example, a command used to print three copies of a file on the lineprinter could be: "PRINT filename/COPIES:3."

synchronous: The performance of a sequence of operations controlled by an external clocking device. Implies that no operation can take place until the previous operation is complete.

synchronous system trap: A system condition that occurs as a result of an error or fault within the executing task.

system device: The device on which the operating system is stored.

system generation: The process of building an operating system on or for a particular hardware configuration with software configuration modifications.

system manager: The person at a computer installation responsible for the overall nature of its operation.

system program: 1. A program that performs system-level functions. 2. Any program that is part of the basic operating system. 3. A system utility program.

system programmer: A person who designs and codes the programs that control the basic operations of a computer system, as opposed to an application programmer.

system UCI: The User Class Identifier (UCI) code in a DSM-11 system that is assigned to the first entry in the system's UCI table. The program and global directories associated with the System UCI are used to contain both system and library programs and globals.

task: In RSX-11 terminology, a load module with special characteristics. In general, any discrete operation performed by a program.

TELENET: The Packet-Switching Network available in the United States.

terminal: An I/O device, such as an LA36 terminal, that includes a keyboard and a display mechanism. In PDP-11 systems, a terminal is used as the primary communication device between a computer system and a person.

time quantum: In timesharing, a unit of time allotted each user by the monitor.

timesharing: A method of allocating CPU time and other computer services to multiple users so that the computer, in effect, processes a number of programs concurrently.

time slice: The period of time allocated by the operating system to process a particular program.

topology: The physical or logical placement of nodes in a computer network.

transaction: A single predefined data processing operation within an application.

transaction processor: A collection of data tables and software capable of processing an application's transactions.

tributary station: A station, other than the control station, on a centralized multipoint data communication system, that can communicate only with the control station when polled or selected by the control station.

turnkey: 1. Pertaining to a computer system and its software that are sold together ready to go. 2. A computer console containing only one control — a power switch to turn the system on and off.

staging: The delay of each update to a file until the end of the transaction instance requesting the update.

unattended operation: The automatic features of a node's operation that permit the transmission and reception of messages on an unattended basis.

unbundling: A practice by which a computer manufacturer does not sell computer equipment and software under one price structure, but sells them separately.

unformatted ASCII: A mode of data transfer in which the low-order seven bits of each byte are transferred. No special formatting of the data occurs or is recognized.

unformatted binary: A mode of data transfer in which all bits of a byte are transferred without regard to their contents.

UNIBUS: The single, asynchronous, high-speed bus structure shared by the PDP-11 processor, its memory, and all of its peripherals.

unmapped system: An RSX-11M or RSX-11S system that does not have a hardware memory management unit available for virtual address relocation.

user class identifier: An identification code that enables a user to gain access to a DSM-11 system to execute programs.

user identification code: The number or set of numbers that serves to distinguish a particular user or collection of files in a multiuser system. The common format for a user identification code is two numbers separated by a comma, enclosed in brackets, e.g., +(3,11+).

user program: An application program.

utility: Any general purpose program included in an operating system to perform common functions.

variable: The symbolic representation of a logical storage location that can contain a value that changes during a discrete processing operation.

virtual address space: A set of memory addresses that is mapped into physical memory addresses by the paging or relocation hardware when a program is executed.

virtual array: A RSTS/E file structure that is logically organized as a dimensioned array.

virtual block: One of a collection of blocks constituting a file (or the memory image of that file). The block is virtual only in that its block number refers to its position relative to other blocks within the file, instead of to its position relative to other blocks on the volume. That is, the virtual blocks of a file are numbered sequentially beginning with one, while their corresponding logical block numbers can be any random list of valid volume-relative block numbers.

volume: A mass-storage medium that can be treated as file-structured data storage.

word: Sixteen binary digits treated as a unit in PDP-11 processor memory.

X.25: A communication protocol created by CCITT that recommends how a computer connects to a packet-switched network. PTTs have accepted X.25 as their standard for Public Packet Switching Networks.

zero a device: To erase all the data stored on a volume and reinitialize the format of the volume.

Index

A

B

C

E

F

J

K

L

M

N

O

Q

R

S

U

V

W

X

Z

PDP-11 Software Handbook

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our handbooks.

What is your general reaction to this handbook? (format, accuracy, completeness, organization, etc.)

What features are most useful?

Does the publication satisfy your needs?

What errors have you found?

Additional comments

Name

Title

Company

Address

City State Zip

EB-28783-41

(staple here)

(staple here)

(please fold here)

No Postage
Necessary if
Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Corporate Marketing Communications
CFO1-2/J48
200 Baker Avenue
West Concord, MA 01742

digital